
Copyright © 2003-2013 Altair Engineering, Inc. All rights reserved.

PBS™, PBS Works™, PBS GridWorks®, PBS Professional®, PBS Analytics™, PBS Catalyst™, e-Compute™, and e-Render™ are trademarks of Altair Engineering, Inc. and are protected under U.S. and international laws and treaties. All other marks are the property of their respective owners.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside ALTAIR and its licensed clients. Information contained herein shall not be decompiled, disassembled, duplicated or disclosed in whole or in part for any purpose. Usage of the software is only as explicitly permitted in the end user software license agreement.

Copyright notice does not imply publication.

For documentation and the PBS Works forums, go to:
Web: www.pbsworks.com
For more information, contact Altair at:
Email: pbssales@altair.com

Technical Support

<table>
<thead>
<tr>
<th>Location</th>
<th>Telephone</th>
<th>e-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>North America</td>
<td>+1 248 614 2425</td>
<td><a href="mailto:pbssupport@altair.com">pbssupport@altair.com</a></td>
</tr>
<tr>
<td>China</td>
<td>+86 (0)21 6117 1666</td>
<td><a href="mailto:es@altair.com.cn">es@altair.com.cn</a></td>
</tr>
<tr>
<td>France</td>
<td>+33 (0)1 4133 0992</td>
<td><a href="mailto:francesupport@altair.com">francesupport@altair.com</a></td>
</tr>
<tr>
<td>Germany</td>
<td>+49 (0)7031 6208 22</td>
<td><a href="mailto:hwsupport@altair.de">hwsupport@altair.de</a></td>
</tr>
<tr>
<td>India</td>
<td>+91 80 66 29 4500</td>
<td><a href="mailto:pbs-support@india.altair.com">pbs-support@india.altair.com</a></td>
</tr>
<tr>
<td>Italy</td>
<td>+39 800 905595</td>
<td><a href="mailto:support@altairengineering.it">support@altairengineering.it</a></td>
</tr>
<tr>
<td>Japan</td>
<td>+81 3 5396 2881</td>
<td><a href="mailto:pbs@altairjp.co.jp">pbs@altairjp.co.jp</a></td>
</tr>
<tr>
<td>Korea</td>
<td>+82 70 4050 9200</td>
<td><a href="mailto:support@altair.co.kr">support@altair.co.kr</a></td>
</tr>
<tr>
<td>Scandinavia</td>
<td>+46 (0) 46 460 2828</td>
<td><a href="mailto:support@altair.se">support@altair.se</a></td>
</tr>
<tr>
<td>UK</td>
<td>+44 (0)1926 468 600</td>
<td><a href="mailto:pbssupport@uk.altair.com">pbssupport@uk.altair.com</a></td>
</tr>
</tbody>
</table>

This document is proprietary information of Altair Engineering, Inc.
# Table of Contents

About PBS Documentation ........................................... ix

1 New Features ......................................................... 1
  1.1 New Features in PBS 12.2 .................................. 1
  1.2 Changes in Previous Releases .......................... 2
  1.3 Deprecations and Removals .............................. 9
  1.4 Backward Compatibility ................................ 12

2 Configuring the Server and Queues .......................... 15
  2.1 The Server ................................................ 15
  2.2 Queues ...................................................... 18

3 Configuring MoMs and Vnodes ................................. 37
  3.1 Vnodes: Virtual Nodes ................................... 37
  3.2 MoMs ......................................................... 43
  3.3 Files and Directories Used by MoM .................... 44
  3.4 MoM and Vnode Configuration Files .................. 46
  3.5 How to Configure MoMs and Vnodes ................. 50
  3.6 Configuring MoM and Vnode Features ................ 57

4 Scheduling .......................................................... 63
  4.1 Chapter Contents ........................................... 63
  4.2 Scheduling Policy Basics ............................... 65
  4.3 Choosing a Policy ........................................ 86
  4.4 The Scheduler .............................................. 100
  4.5 Using Queues in Scheduling ............................ 114
  4.6 Scheduling Restrictions and Caveats ................ 115
  4.7 Errors and Logging ....................................... 116
  4.8 Scheduling Tools ........................................ 116
# Table of Contents

## 5 PBS Resources

5.1 Introduction ......................................................... 287  
5.2 Chapter Contents ........................................................ 288  
5.3 Terminology ............................................................ 289  
5.4 Categories of Resources ............................................... 293  
5.5 Resource Types .......................................................... 298  
5.6 Behavior of Resources .................................................. 298  
5.7 How to Set Resource Values ............................................ 300  
5.8 Overview of Ways Resources Are Used ............................... 303  
5.9 Resources Allocated to Jobs and Reservations ....................... 304  
5.10 Using Resources to Track and Control Allocation ................. 314  
5.11 Using Resources to Prioritize Jobs .................................... 317  
5.12 Using Resources to Restrict Server, Queue Access ................ 318  
5.13 Custom Resources ..................................................... 319  
5.14 Managing Resource Usage .............................................. 366  
5.15 Where Resource Information Is Kept .................................. 400  
5.16 Viewing Resource Information ......................................... 406  
5.17 Resource Recommendations and Caveats ............................ 408  
5.18 Advice and Caveats ..................................................... 408

## 6 Hooks

6.1 Chapter Contents ......................................................... 413  
6.2 Introduction to Hooks .................................................... 416  
6.3 Terminology ............................................................ 417  
6.4 Prerequisites and Requirements for Hooks ............................ 418  
6.5 Updating Hooks and Custom Resources ............................... 419  
6.6 Simple How-to for Writing Hooks ..................................... 420  
6.7 Examples of Using Hooks ............................................... 426  
6.8 When, as Whom, and Where Hooks Run ............................... 430  
6.9 What Hooks Do ........................................................... 434  
6.10 Creating Hooks .......................................................... 439  
6.11 Configuring Hooks ....................................................... 446  
6.12 Viewing Hook Information .............................................. 452  
6.13 Interface to Hooks ....................................................... 454  
6.14 Hook Examples ............................................................ 536  
6.15 Advice and Caveats ....................................................... 548  
6.16 Python Modules and PBS ................................................ 566  
6.17 Errors, Logging, and Troubleshooting .................................. 569  
6.18 Attributes, Parameters Affecting Hooks ............................. 594  
6.19 See Also ........................................................................ 594
# Table of Contents

## 7 Provisioning
- 7.1 Introduction ................................................. 597
- 7.2 Definitions .................................................. 597
- 7.3 How Provisioning Can Be Used ......................... 598
- 7.4 How Provisioning Works ................................. 599
- 7.5 Configuring Provisioning ................................. 609
- 7.6 Viewing Provisioning Information .................... 616
- 7.7 Requirements and Restrictions ....................... 619
- 7.8 Defaults and Backward Compatibility .............. 622
- 7.9 Example Scripts ........................................... 622
- 7.10 Advice and Caveats ...................................... 637
- 7.11 Errors and Logging ...................................... 639

## 8 Security
- 8.1 Configurable Features ..................................... 643
- 8.2 Setting User Roles .......................................... 644
- 8.3 Using Access Control ..................................... 647
- 8.4 Restricting Execution Host Access .................. 668
- 8.5 Logging Security Events ............................... 670
- 8.6 Changing the PBS Service Account Password ..... 673
- 8.7 Paths and Environment Variables ..................... 674
- 8.8 File and Directory Permissions ......................... 675
- 8.9 Authentication & Authorization ....................... 676
- 8.10 Root-owned Jobs ......................................... 680
- 8.11 User Passwords ............................................ 680
- 8.12 Windows Caveats .......................................... 684
- 8.13 Windows XP SP2 Firewall .............................. 684
- 8.14 Windows Requirement for cmd Prompt .......... 684

## 9 Making Your Site More Robust
- 9.1 Robustness .................................................. 685
- 9.2 Failover ...................................................... 686
- 9.3 Checkpoint and Restart .................................. 711
- 9.4 Preventing Communication and Timing Problems . 733
- 9.5 Reservation Fault Tolerance ............................ 743
- 9.6 Preventing File System Problems .................... 746
- 9.7 Built-in Robustness ........................................ 747
# Table of Contents

## 10 Integrations

10.1 Integration with MPI ............................................. 749
10.2 Support for IBM AIX ........................................... 771
10.3 Support for Cray Systems ..................................... 777
10.4 Support for SGI .................................................. 808
10.5 Support for Globus .............................................. 818
10.6 Support for Hyper-Threading ................................. 818

## 11 Managing Jobs

11.1 Routing Jobs ....................................................... 821
11.2 Allocating Resources to Jobs ................................ 821
11.3 Grouping Jobs By Project ...................................... 823
11.4 Job Prologue and Epilogue ..................................... 825
11.5 UNIX Shell Invocation ........................................... 831
11.6 Job’s Life Cycle ................................................... 831
11.7 When Job Attributes are Set .................................. 832
11.8 Job Termination .................................................... 835
11.9 Job Exit Codes ..................................................... 839
11.10 Rerunning or Requeueing a Job .............................. 842
11.11 Job IDs ............................................................ 843
11.12 Where to Find Job Information ............................... 843
11.13 Job Directories ..................................................... 844
11.14 The Job’s Lifecycle .............................................. 849
11.15 Managing Job History .......................................... 852
11.16 Environment Variables .......................................... 857
11.17 Adjusting Job Running Time .................................. 857
11.18 Managing Number of Run Attempts ......................... 857
11.19 Allowing Interactive Jobs on Windows ..................... 858
# Table of Contents

## 12 Administration

12.1 The PBS Configuration File ................................................. 859
12.2 Environment Variables ........................................................ 862
12.3 The Accounting Log ............................................................. 864
12.4 Event Logging ........................................................................ 867
12.5 Using the UNIX *syslog* Facility .......................................... 875
12.6 Managing Machines .............................................................. 876
12.7 Managing the Data Service .................................................... 877
12.8 Enabling Passwordless Authentication ..................................... 880
12.9 Setting File Transfer Mechanism .......................................... 881
12.10 Temporary File Location for PBS Components ....................... 890
12.11 Administration Caveats ....................................................... 892

## 13 Problem Solving

13.1 Debugging PBS ..................................................................... 893
13.2 Server Host Bogs Down After Startup .................................... 893
13.3 Finding PBS Version Information ........................................ 894
13.4 Troubleshooting and Hooks .................................................. 894
13.5 Directory Permission Problems .......................................... 895
13.6 Common Errors ..................................................................... 895
13.7 Errors on Windows ............................................................... 899
13.8 Troubleshooting PBS Licenses .............................................. 905
13.9 Security-related Problems .................................................... 907
13.10 Time Zone Problems ............................................................ 910
13.11 Job Comments for Problem Jobs ......................................... 910
13.12 Getting Help ....................................................................... 911

### Appendix A: License Agreement

913

### Appendix B: Third-party Legal Notices

923

### Index

931
About PBS Documentation

Where to Keep the Documentation
To make cross-references work, put all of the PBS guides in the same directory.

What is PBS Professional?
PBS is a workload management system that provides a unified batch queuing and job management interface to a set of computing resources.

The PBS Professional Documentation
The documentation for PBS Professional includes the following:

PBS Professional Administrator’s Guide:
Provides the PBS administrator with the information required to configure and manage PBS Professional (PBS).

PBS Professional Quick Start Guide:
Provides a quick overview of PBS Professional installation and license file generation.

PBS Professional Installation & Upgrade Guide:
Contains information on installing and upgrading PBS Professional.

PBS Professional User’s Guide:
Covers user commands and how to submit, monitor, track, delete, and manipulate jobs.

PBS Professional Programmer’s Guide:
Discusses the PBS application programming interface (API).
About PBS Documentation

PBS Professional Reference Guide:
Contains PBS reference material.

PBS Manual Pages:
Describe PBS commands, resources, attributes, APIs

Ordering Software and Publications
To order additional copies of this manual and other PBS publications, or to purchase additional software licenses, contact your Altair sales representative. Contact information is included on the copyright page of this book.

Document Conventions
PBS documentation uses the following typographic conventions:

**abbreviation**
The shortest acceptable abbreviation of a command or subcommand is underlined.

**command**
Commands such as `qmgr` and `scp`

**input**
Command-line instructions

**manpage(x)**
File and path names. Manual page references include the section number in parentheses appended to the manual page name.

**formats**
Formats

**Attributes**
Attributes, parameters, objects, variable names, resources, types

**Values**
Keywords, instances, states, values, labels

**Definitions**
Terms being defined
About PBS Documentation

Output
  Output or example code
  File contents
About PBS Documentation
Chapter 1

New Features

This chapter briefly lists new features by release, with the most recent listed first. This chapter also lists deprecated elements, such as options, keywords, etc.

The Release Notes included with this release of PBS Professional list all new features in this version of PBS Professional, and any warnings or caveats. Be sure to review the Release Notes, as they may contain information that was not available when this book was written.

The PBS Professional manual pages that were reproduced in this guide are available in the PBS Professional Reference Guide or as UNIX man pages. They have been removed from this book to save space.

1.1 New Features in PBS 12.2

1.1.1 Command Line Editing in qmgr

The qmgr command provides a history and allows you to edit command lines. See “Reusing and Editing the qmgr Command Line” on page 152 of the PBS Professional Reference Guide.

1.1.2 Interactive Jobs Available under Windows

Job submitters can run interactive jobs under Windows. See "Running Your Job Interactively", on page 176 of the PBS Professional User’s Guide.
1.1.3 Job Run Count is Writable

Job submitters and administrators can set the value of a job’s run count. See section 11.18, “Managing Number of Run Attempts”, on page 857 and “Controlling Number of Times Job is Re-run”, on page 174 of the PBS Professional User’s Guide.

1.1.4 runjob Hook can Modify Job Attributes

The runjob hook can modify a job’s attributes and resources. See section 6.13.5, “Using Attributes and Resources in Hooks”, on page 456.

1.1.5 Jobs can be Suspended under Windows

You can suspend and resume a job under Windows.

1.1.6 Configuration of Directory for PBS Component Temporary Files

You can configure the root directory where you want PBS components to put their temporary files. See section 12.10, “Temporary File Location for PBS Components”, on page 890.

1.2 Changes in Previous Releases

1.2.1 Execution Event and Periodic Hooks (12.0)

You can write hooks that run at the execution host when the job reaches the execution host, when the job starts, ends, is killed, and is cleaned up. You can also write hooks that run periodically on all execution hosts. See Chapter 6, "Hooks", on page 413.

1.2.2 Shrink-to-fit Jobs (12.0)

PBS allows users to specify a variable running time for jobs. Job submitters can specify a walltime range for jobs where attempting to run the job in a tight time slot can be useful. Administrators can convert non-shrink-to-fit jobs into shrink-to-fit jobs in order to maximize machine use. See "Adjusting Job Running Time", on page 161 of the PBS Professional User’s Guide and section 4.8.41, “Using Shrink-to-fit Jobs”, on page 263.
1.2.3 **PBS Supports Socket Licensing (11.3)**

PBS lets you use socket licenses to license hosts. See “Overview of Licensing for PBS Jobs” on page 89 in the PBS Professional Installation & Upgrade Guide.

1.2.4 **Deleting Job History (11.3)**

You can delete job histories. See section 11.15.8, “Deleting Moved Jobs and Job Histories”, on page 856.

1.2.5 **Managing Resource Usage by Project (11.2)**

You can set resource usage limits for projects, at the server and queue. You can set limits for the amount of each resource being used, or for the number of jobs. Jobs have a new attribute called *project*. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367.

1.2.6 **Support for Accelerators on Cray (11.2)**


1.2.7 **PBS Daemons Protected from OOM Killer (11.2)**

PBS daemons are protected from being terminated by an OOM killer. See section 9.7.1, “OOM Killer Protection”, on page 747.

1.2.8 **PBS Supports X Forwarding for Interactive Jobs (11.2)**


1.2.9 **Support for Interlagos on Cray (11.1)**

You can allow users to request vnodes that have Interlagos hardware. See section 10.3.7.14, “Allowing Users to Request Interlagos Hardware”, on page 795.
New Features

1.2.10  Improved Cray Integration (11.0)

PBS is more tightly integrated with Cray systems. You can use the PBS select and place language when submitting Cray jobs. See section 10.3, “Support for Cray Systems”, on page 777.

1.2.11  Vnode Access for Hooks (11.0)

Hooks have access to vnode attributes and resources. See Chapter 6, "Hooks", on page 413.

1.2.12  Enhanced Job Placement (11.0)

PBS allows job submitters to scatter chunks by vnode in addition to scattering by host. PBS also allows job submitters to reserve entire hosts via a job’s placement request. See "Specifying Job Placement", on page 89 of the PBS Professional User’s Guide.

1.2.13  Choice in PBS service account Name (11.0)

Under Windows, the PBS service account used to run PBS daemons can have any name. See “The PBS Service Account” on page 18 in the PBS Professional Installation & Upgrade Guide and “The PBS service account for Standalone Environments” on page 21 in the PBS Professional Installation & Upgrade Guide.

1.2.14  Change of Licensing Method (11.0)

As of 11.0, PBS is licensed using a new Altair license server. See “Licensing” on page 89 in the PBS Professional Installation & Upgrade Guide.

1.2.15  Change in Data Management (11.0)

PBS uses a new data service. See section 12.7, “Managing the Data Service”, on page 877.

1.2.16  Choice in Job Requeue Timeout (11.0)

You can choose how long the job requeue process should be allowed to run. See section 9.4.3, “Setting Job Requeue Timeout”, on page 737.
1.2.17 Backfilling Around Top $N$ Jobs (10.4)

PBS can backfill around the most deserving jobs. You can configure the number of jobs PBS backfills around. See section 4.8.3, “Using Backfilling”, on page 124.

1.2.18 Estimating Job Start Times (10.4)

PBS can estimate when jobs will run, and which vnodes each job will use. See section 4.8.15, "Estimating Job Start Time", on page 161.

1.2.19 Unified Job Submission (10.4)

PBS allows users to submit jobs using the same scripts, whether the job is submitted on a Windows or UNIX/Linux system. See "Python Job Scripts", on page 25 of the PBS Professional User’s Guide.

1.2.20 Provisioning (10.2)

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application. See Chapter 7, "Provisioning", on page 597.

1.2.21 New Hook Type (10.2)

PBS has a new hook type which can be triggered when a job is to be run. See “Hooks” on page 413.

1.2.22 New Scheduler Attribute (10.2)

PBS allows the administrator to set the scheduler’s cycle time using the new sched_cycle_length scheduler attribute. See the pbs_sched_attributes(7B) manual page.

1.2.23 Walltime as Checkpoint Interval Measure (10.2)

PBS allows a job to be checkpointed according to its walltime usage. See the pbs_job_attributes(7B) manual page.
Chapter 1  New Features

1.2.24 Employing User Space Mode on IBM InfiniBand Switches (10.2)

PBS allows users submitting POE jobs to use InfiniBand switches in User Space mode. See section 10.2, “Support for IBM AIX”, on page 771.

1.2.25 Managing Resource Usage (10.1)

You can set separate limits for resource usage by individual users, individual groups, generic users, generic groups, and the total used. You can limit the amount of resources used, and the number of running and queued jobs. These limits can be defined separately for each queue and for the server. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367. These new limits are incompatible with the limit attributes existing before Version 10.1.

1.2.26 Managing Job History (10.1)

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used. PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server. See section 11.15, “Managing Job History”, on page 852.

1.2.27 Reservation Fault Tolerance (10.1)

PBS attempts to reconfirm reservations for which associated vnodes have become unavailable. See section 9.5, “Reservation Fault Tolerance”, on page 743.

1.2.28 Checkpoint Support via Epilogue (10.1)

Checkpointed jobs can be requeued if the epilogue exits with a special value. See section 9.3.7.3, “Requeueing via Epilogue”, on page 730.
1.2.29 Hooks (10.0)

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches or workarounds, and extends the capabilities of PBS, without the need to modify source code. See section “Hooks”, on page 413.

1.2.30 Versioned Installation (10.0)

PBS is now automatically installed in versioned directories. For most platforms, different versions of PBS can coexist, and upgrading is simplified. See Chapter 3, "Installation", on page 29 and Chapter 6, "Upgrading PBS Professional", on page 111 in the PBS Professional Installation and Upgrade Guide.

1.2.31 Resource Permissions for Custom Resources (9.2)

You can set permissions on custom resources so that they are either invisible to users or cannot be requested by users. This also means that users cannot modify a resource request for those resources via qalter. See section 5.14.2.6, “Resource Permission Flags”, on page 328.

1.2.32 Extension to Job Sorting Formula (9.2)

The job sorting formula has been extended to include parentheses, exponentiation, division, and unary plus and minus. See section 4.8.3, “Using Backfilling”, on page 124.

1.2.33 Eligible Wait Time for Jobs (9.2)

A job that is waiting to run can be accruing “eligible time”. Jobs can accrue eligible time when they are blocked due to a lack of resources. This eligible time can be used in the job sorting formula. Jobs have two new attributes, eligible_time and accrue_type, which indicates what kind of wait time the job is accruing. See section 4.8.13, “Eligible Wait Time for Jobs”, on page 155.

1.2.34 Job Staging and Execution Directories (9.2)

PBS now provides per-job staging and execution directories. Jobs have new attributes sandbox and jobdir, the MoM has a new option $jobdir_root, and there is a new environment variable called PBS_JOBDIR. If the job’s sandbox attribute is set to PRIVATE, PBS cre-
Chapter 1  New Features

ates a job-specific staging and execution directory. If the job’s sandbox attribute is unset or is set to HOME, PBS uses the user’s home directory for staging and execution, which is how previous versions of PBS behaved. If MoM’s $jobdir_root is set to a specific directory, that is where PBS will create job-specific staging and execution directories. If MoM’s $jobdir_root is unset, PBS will create the job-specific staging and execution directory under the user’s home directory. See section 11.13.1, “The Job’s Staging and Execution Directories”, on page 844.

1.2.35  Standing Reservations (9.2)

PBS now provides both advance and standing reservation of resources. A standing reservation is a reservation of resources for specific recurring periods of time. See section 4.8.37, “Advance and Standing Reservations”, on page 249.

1.2.36  New Server Attribute for Job Sorting Formula (9.1)

The new server attribute "job_sort_formula" is used for sorting jobs according to a site-defined formula. See section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.

1.2.37  Change to sched_config (9.1)

The default for job_sort_key of “cput” is commented out in the default sched_config file. It is left in as a usage example.

1.2.38  Change to Licensing (9.0)

PBS now depends on an Altair license server that will hand out licenses to be assigned to PBS jobs. See “Licensing” on page 89 in the PBS Professional Installation & Upgrade Guide. PBS Professional versions 8.0 and below will continue to be licensed using the proprietary licensing scheme.

1.2.39  Installing With Altair Licensing (9.0)

If you will use floating licenses, we recommend that you install and configure the Altair license server before installing and configuring PBS. PBS starts up faster. See “Overview of Installation” on page 29 in the PBS Professional Installation & Upgrade Guide.
1.2.40  Unset Host-level Resources Have Zero Value (9.0)

An unset numerical resource at the host level behaves as if its value is zero, but at the server or queue level it behaves as if it were infinite. An unset string or string array resource cannot be matched by a job’s resource request. An unset boolean resource behaves as if it is set to “False”. See section 4.8.28.7, “Matching Unset Resources”, on page 199.

1.2.41  Better Management of Resources Allocated to Jobs (9.0)

The resources allocated to a job from vnodes will not be released until certain allocated resources have been freed by all MoMs running the job. The end of job accounting record will not be written until all of the resources have been freed. The “end” entry in the job end (‘E’) record will include the time to stage out files, delete files, and free the resources. This will not change the recorded “walltime” for the job.

1.2.42  Support for Large Page Mode on AIX (9.0)

PBS Professional supports Large Page Mode on AIX. No additional steps are required from the PBS administrator.

1.3  Deprecations and Removals

The -a alarm option to pbs_sched is deprecated, and is replaced with the sched_cycle_length scheduler attribute.

The sort_priority option to job_sort_key is deprecated and is replaced with the job_priority option.

The -lnodes=nodespec form is replaced by the -l select= and -l place= statements.

The nodes resource is no longer used.

The -l resource=rescspec form is replaced by the -l select= statement.

The time-shared node type is no longer used, and the .ts suffix is obsolete.

The cluster node type is no longer used.

The resource arch is only used inside of a select statement.

The resource host is only used inside of a select statement.
Chapter 1  

**New Features**

The **nodect** resource is obsolete. The **ncpus** resource should be used instead. Sites which currently have default values or limits based on **nodect** should change them to be based on **ncpus**.

The **neednodes** resource is obsolete.

The **ssinodes** resource is obsolete.

**Properties** are replaced by boolean resources.

The **-a** option to the **qselect** command is deprecated.

The **-Wdelay=nnnn** option to the **qdel** command is deprecated.

The **-c** and **-d** options to the **pbsnodes** command are deprecated.

The **-c** and **-d** options to **pbsnodes** are removed.

The **memreserved** MoM configuration option is deprecated.

The **pbs_tclapi pbsrescquery** command is deprecated.

The **pbs_rescquery** command is deprecated.

The **sync_time** scheduler configuration option is deprecated.

The Cray **mpp** syntax is deprecated with PBS version 11. Requesting the **mpp** resources in a **qsub**, **pbs_rsub**, or **qalter** command is deprecated.

- The following resources are deprecated:
  - **mppwidth**
  - **mppdepth**
  - **mppnppn**
  - **mppmem**
  - **mpparch**
  - **mpphost**
  - **mpplabels**
  - **mppnodes**

- PBS does not support server or queue level **mpp** defaults. The following are depre-
New Features

- PBS does not support mpp* minima or maxima for server and queues. The following are deprecated:
  
  resources_default.mppwidth
  resources_default.mppdepth
  resources_default.mppnppn
  resources_default.mppmem
  resources_default.mpparch
  resources_default.mpphost
  resources_default.mpplabels
  resources_default.mppnodes

  resources_min.mppwidth
  resources_min.mppdepth
  resources_min.mppnppn
  resources_min.mppmem
  resources_min.mpparch
  resources_min.mpphost
  resources_min.mpplabels
  resources_min.mppnodes
  resources_max.mppwidth
  resources_max.mppdepth
  resources_max.mppnppn
  resources_max.mppmem
  resources_max.mpparch
  resources_max.mpphost
  resources_max.mpplabels
  resources_max.mppnodes

The **pbs_license_file_location** server attribute is deprecated and replaced by **pbs_license_info**.

The **configrm()** resource monitor API call is deprecated.
Support in PBS for CSA on SGI systems is removed.
Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus (11.3).
Support for LAM MPI 6.5.9 is deprecated (12.0).
In version 12.0, PBS uses Python 2.5. PBS will use a newer version of Python in some subsequent release, so support for Python 2.5 is deprecated. (12.0).
The pbs-report command is deprecated, and will be moved to the unsupported directory in the next release.
The sort_queues scheduler parameter is deprecated. (12.2).
The smp_cluster_dist scheduler parameter is deprecated. (12.2).
Support for HPCBP jobs is removed (12.2).

1.4 Backward Compatibility

1.4.1 New and Old Resource Usage Limits Incompatible


1.4.2 Job Dependencies Affected By Job History

Enabling job history changes the behavior of dependent jobs. If a job j1 depends on a finished job j2 for which PBS is maintaining history than j1 will go into the held state. If job j1 depends on a finished job j3 that has been purged from the historical records than j1 will be rejected just as in previous versions of PBS where the job was no longer in the system.

1.4.3 PBS path information no longer saved in AUTOEXEC.BAT

Any value for PATH saved in AUTOEXEC.BAT may be lost after installation of PBS. If there is any path information that needs to be saved, AUTOEXEC.BAT must be edited by hand after the installation of PBS. PBS path information is no longer saved in AUTOEXEC.BAT.
1.4.4 OS-level Checkpointing Not Supported

PBS does not directly support OS-level checkpointing. PBS supports checkpointing using site-supplied methods. See section 9.3, “Checkpoint and Restart”, on page 71.
Chapter 1  New Features
Chapter 2

Configuring the Server and Queues

This chapter describes how to configure the server and any queues.

2.1 The Server

2.1.1 Configuring the Server

You configure the server by setting server attributes via the `qmgr` command:

```
Qmgr: set server <attribute> = <value>
```

For a description of the server attributes, see “Server Attributes” on page 316 of the PBS Professional Reference Guide.

For a description of the `qmgr` command, see “qmgr” on page 150 of the PBS Professional Reference Guide.
2.1.2 Default Server Configuration

The default configuration from the binary installation sets the default server settings. An example server configuration is shown below:

```
qmgr
Qmgr: print_server
#
# Create queues and set their attributes.
# Create and define queue workq
#
create_queue workq
set_queue workq queue_type = Execution
set_queue workq enabled = True
set_queue workq started = True
#
# Set server attributes.
#
set_server scheduling = True
set_server default_queue = workq
set_server log_events = 511
set_server mail_from = adm
set_server query_other_jobs = True
set_server resources_default.ncpus = 1
set_server scheduler_iteration = 600
set_server resv_enable = True
set_server node_fail_requeue = 310
set_server max_array_size = 10000
set_server default_chunk.ncpus = 1
```

2.1.3 The Server’s Node File

The server creates a file of the nodes managed by PBS. This node file is written only by the Server. On startup each MoM sends a time-stamped list of her known vnodes to the Server. The Server updates its information based on that message. If the time stamp on the vnode list is newer than what the Server recorded before in the node file, the Server will create any
vnodes which were not already defined. If the time stamp in the MoM’s message is not newer, then the Server will not create any missing vnodes and will log an error for any vnodes reported by MoM but not already known.

Whenever new vnodes are created, the Server sends a message to each MoM with the list of MoMs and each vnode managed by the MoMs. The Server will only delete vnodes when they are explicitly deleted via `qmgr`.

This is different from the node file created for each job. See "The Job’s Node File", on page 106 of the PBS Professional User’s Guide.

### 2.1.4 Server Configuration Attributes

See “Server Attributes” on page 316 of the PBS Professional Reference Guide for a table of server attributes.

### 2.1.5 Recording Server Configuration

If you wish to record the configuration of a PBS Server for re-use later, you may use the `print` subcommand of `qmgr(8B)`. For example,

```
qmgr -c "print server" > /tmp/server.out
qmgr -c "print node @default" > /tmp/nodes.out
```

will record in the file `/tmp/server.out` the `qmgr` subcommands required to recreate the current configuration including the queues. The second file generated above will contain the vnodes and all the vnode properties. The commands could be read back into `qmgr` via standard input:

```
qmgr < /tmp/server.out
qmgr < /tmp/nodes.out
```

### 2.1.6 Support for Globus

Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus. The Globus MoM is no longer available.
2.1.7 Configuring the Server for Licensing

The PBS server must be configured for licensing. You must set the location where PBS will look for the license file and/or license server(s), by setting the server attribute pbs_license_info. The other server licensing attributes have defaults, but you may wish to set them as well. See “Configuring PBS for Licensing” on page 93 in the PBS Professional Installation & Upgrade Guide.

You may also wish to have redundant license servers. See the Altair License Management System Installation and Operations Guide, available at www.pbsworks.com.

2.1.8 Configuring Mail

You can configure the account that is used as the address to both send and receive administrative mail. These are the same account. For example, when failover occurs, an email is sent to and from the account defined in the server’s mail_from attribute, saying that failover has occurred.

Use the qmgr command to set the mail_from server attribute to an address that is monitored regularly:

   Qmgr: s server mail_from=<address>

You cannot configure which mail server PBS uses. PBS uses the default mail server. On UNIX/Linux, this is /usr/lib/sendmail.

On Windows, PBS uses sendmail on the host specified in the server’s mail_from attribute. For example, if you set mail_from to admin_acct@host1.example.com, PBS uses sendmail on host1.

2.1.8.1 Mail Caveats

If you do not set the mail_from attribute on Windows, PBS will not be able to send mail.

2.2 Queues

When a job is submitted to PBS and accepted, it is placed in a queue. Despite the fact that the name implies first-in, first-out ordering of jobs, this is not the case. Job submission order does not determine job execution order. See Chapter 4, "Scheduling", on page 63.
You can create different queues for different purposes: queues for certain kinds of jobs, queues for specific groups, queues for specific nodes, etc. You can tell PBS how to automatically route jobs into each queue. PBS has a default execution queue named workq, where jobs are placed when no queue is requested. You can specify which queue should be the default. See section 2.2.14, “Specifying Default Queue”, on page 34.

2.2.1 Kinds of Queues

2.2.1.1 Execution and Routing Queues

There are two main types of PBS queues: routing and execution.

- A routing queue is used only to move jobs to other queues. These destination queues can be routing or execution queues, and can be located at different PBS servers.
- An execution queue is used as the home for a waiting or running job. A job must reside in an execution queue to be eligible to run. The job remains in the execution queue during the time it is running.

For more information on creating and using routing queues, see section 2.2.6, “Routing Queues”, on page 24.

For information on configuring routing queues and failover, see section 9.2.6.1, “Configuring Failover to Work with Routing Queues”, on page 707.

2.2.1.2 Available Kinds of Queues

PBS supplies the following kinds of execution and routing queues:

<table>
<thead>
<tr>
<th>Kind of Queue</th>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing queues</td>
<td>Used for moving jobs to another queue</td>
<td>See section 2.2.6, “Routing Queues”, on page 24</td>
</tr>
</tbody>
</table>
Table 2-1: Kinds of Queues

<table>
<thead>
<tr>
<th>Kind of Queue</th>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution queues</td>
<td>Created for reservation.</td>
<td>See section 2.2.5.2.iv, “Reservation Queues”, on page 23</td>
</tr>
<tr>
<td>Reservation queues</td>
<td>Holds jobs that run only during dedicated time.</td>
<td>See section 2.2.5.2.i; “Dedicated Time Queues”, on page 23</td>
</tr>
<tr>
<td>Dedicated time queues</td>
<td>Holds jobs that run only during primetime.</td>
<td>See section 2.2.5.2.ii, “Prime-time and Non-Primetime Queues”, on page 23</td>
</tr>
<tr>
<td>Primetime queues</td>
<td>Holds jobs that run only during non-primetime.</td>
<td>See section 2.2.5.2.ii, “Prime-time and Non-Primetime Queues”, on page 23</td>
</tr>
<tr>
<td>Non-primetime queues</td>
<td>Queue with no dedicated time or primetime restric-</td>
<td>See section 2.2.5.2.iii, “Anytime Queues”, on page 23</td>
</tr>
<tr>
<td>Anytime queues</td>
<td>tion</td>
<td></td>
</tr>
<tr>
<td>Express queues</td>
<td>High-priority queue; priority is set to the level signifying that it is an express queue</td>
<td>See section 2.2.5.3.i, “Express Queues”, on page 24</td>
</tr>
<tr>
<td>Anti-express queue</td>
<td>Low-priority queue designed for work that should run only when no other jobs need the resources</td>
<td>See section 4.8.1, “Anti-Express Queues”, on page 120</td>
</tr>
</tbody>
</table>

2.2.2 Basic Queue Use

The simplest form of PBS uses just one queue. The queue is an execution queue named *workq*. This queue is always created, enabled, and started for you during installation. After a basic installation, this queue is ready to hold jobs submitted by users.
2.2.3 Creating Queues

To create a queue, use the `qmgr` command to create it and set its `queue_type` attribute:

```
Qmgr: create queue <queue name>
Qmgr: set queue <queue_name> queue_type = <execution or route>
```

For example, to create an execution queue named `exec_queue`, set its type, start it, and enable it:

```
Qmgr: create queue exec_queue
Qmgr: set queue exec_queue queue_type = execution
Qmgr: set queue exec_queue enabled = True
Qmgr: set queue exec_queue started = True
```

Now we will create a routing queue, which will send jobs to our execution queue:

```
Qmgr: create queue routing_queue
Qmgr: set queue routing_queue queue_type = route
Qmgr: set queue routing_queue route_destinations = exec_queue
```

2.2.4 Enabling, Disabling, Starting, and Stopping Queues

When you enable a queue, you allow it to accept jobs, meaning that jobs can be enqueued in the queue. When you disable a queue, you disallow it from accepting jobs. Queues are disabled by default. You enable a queue by setting its `enabled` attribute to `True`:

```
Qmgr: set queue <queue name> enabled = True
```

When you start a queue, you allow the jobs in the queue to be executed. Jobs are selected to be run according to the scheduling policy. When you stop a queue, you disallow jobs in that queue from running, regardless of scheduling policy. Queues are stopped by default. You start a queue by setting its `started` attribute to `True`:

```
Qmgr: set queue <queue name> started = True
```

2.2.5 Execution Queues

Execution queues are used to run jobs; jobs must be in an execution queue in order to run. PBS does not route from execution queues.
2.2.5.1 Where Execution Queues Get Their Jobs

By default, PBS allows jobs to be moved into execution queues via the qmove command, by hooks, from routing queues, and by being submitted to execution queues. You can specify that an execution queue should accept only those jobs that are routed from a routing queue by PBS, by setting the queue’s from_route_only attribute to True:

```
Qmgr: set queue <queue name> from_route_only = True
```

2.2.5.2 Execution Queues for Specific Time Periods

PBS provides a mechanism that allows you to specify that the jobs in an execution queue can run only during specific time periods. PBS provides a different kind of execution queue for each kind of time period. The time periods you can specify are the following:

**Advance or Standing Reservations**
You can create an advance or standing reservation. An advance reservation is a reservation for specified resources for a specified time period with a defined beginning and end. A standing reservation is a series of recurring advance reservations.

**Dedicated time**
Dedicated time is a period of time with a defined beginning and end. You can define multiple dedicated times.

**Primetime**
Primetime is a recurring time period with a defined beginning and end. You can define primetime to be different for each day of the week.

**Non-primetime**
Non-primetime is a recurring time period with a defined beginning and end. Non-primetime begins when primetime ends, and vice versa.

**Holidays**
Holidays are dates defined in the PBS_HOME/sched_priv/holidays file. PBS has a default set of holidays, and you can define your own holidays. Holiday time is treated like non-primetime, meaning jobs in non-primetime queues run during holiday time.

**Anytime queue**
The term “anytime queue” means a queue that is not a primetime or a non-primetime queue.
2.2.5.2.i Dedicated Time Queues

The jobs in a dedicated time execution queue can run only during dedicated time. Dedicated
time is defined in PBS_HOME/sched_priv/dedicated_time. See section 4.8.10, “Dedicated Time”, on page 153.

To specify that a queue is a dedicated time queue, you prefix the queue name with the dedi-
cated time keyword. This keyword defaults to “ded”, but can be defined in the
dedicated_prefix scheduler parameter in PBS_HOME/sched_priv/sched_config.
See “dedicated_prefix” on page 283 of the PBS Professional Reference Guide.

2.2.5.2.ii Primetime and Non-Primetime Queues

The jobs in a primetime queue run only during primetime, and the jobs in a non-primetime
queue run only during non-primetime. Primetime and non-primetime are defined in
PBS_HOME/sched_priv/holidays. See section 4.8.34, “Using Primetime and Holi-
days”, on page 241.

To specify that a queue is a primetime or non-primetime queue, you prefix the queue name
with the primetime or non-primetime keyword. For primetime, this keyword defaults to
“p_”, and for non-primetime, the keyword defaults to “np_”, but these can be defined in the
primetime_prefix and nonprimetime_prefix scheduler parameters in PBS_HOME/
sched_priv/sched_config. See “Scheduler Parameters” on page 281 of the PBS Pro-
fessional Reference Guide.

2.2.5.2.iii Anytime Queues

An anytime queue is a queue whose jobs can run at any time. An anytime queue is simply a
queue that is not a dedicated time, primetime, or non-primetime queue.

2.2.5.2.iv Reservation Queues

When the pbs_rsub command is used to create a reservation or to convert a job into a reser-
vation job, PBS creates a reservation queue. Jobs in the queue run only during the reserva-

2.2.5.3 Prioritizing Execution Queues

You can set the priority of each execution queue as compared to the other queues in this com-
plex by specifying a value for the priority queue attribute:

Qmgr: set queue <queue name> priority = <value>

A higher value for priority means the queue has greater priority. There is no limit to the prior-
ity that you can assign to a queue, however it must fit within integer size. See “Queue
Attributes” on page 354 of the PBS Professional Reference Guide.
Chapter 2  Configuring the Server and Queues

For how queue priority is used in scheduling, see section 4.8.36, “Queue Priority”, on page 248.

2.2.5.3.i  Express Queues

A queue is an express queue if its priority is greater than or equal to the value that defines an express queue. This value is set in the preempt_queue_prio parameter in PBS_HOME/sched_priv/sched_config. The default value for preempt_queue_prio is 150.

You do not need to set by_queue or sort_queues to True in order to use express queues.

For how express queues can be used, see section 4.8.17, “Express Queues”, on page 169.

2.2.6  Routing Queues

A routing queue is used only to route jobs; jobs cannot run from a routing queue. A routing queue has the following properties:

• Can route to multiple destinations
• Tries destinations in round-robin fashion, in the order listed
• Can route to execution queues
• Can route to other routing queues
• Can route to queues in other complexes (at other servers)

Destinations can be specified in the following ways:

```plaintext
route_destinations = Q1
route_destinations = Q1@Server1
route_destinations = "Q1, Q2@Server1, Q3@Server2"
route_destinations += Q1
route_destinations += "Q4, Q5@Server3"
```

2.2.6.1  How Routing Works

Whenever a job enters a routing queue, PBS immediately attempts to route the job to a destination queue. The result is one of the following:

• The job is routed to one of the destination queues.
• The attempt to route is permanently rejected by each destination, and the job is deleted.
• Every destination rejects the job, but at least one rejection is temporary. In this case, the destination is tried again later.
If there are multiple routing queues containing jobs to be routed, the routing queues are processed in the order in which they are displayed in the output of a `qstat -Q` command.

When PBS routes a job, it tries each destination in the order listed. The job’s destination is the first queue that accepts it.

Queue priority does not play a role in routing jobs.

### 2.2.6.2 Requirements for Routing Queues

- A routing queue’s destination queues must be created before being specified in the routing queue’s `route_destinations` attribute.
- A routing queue’s `route_destinations` attribute must be specified before enabling and starting the routing queue.

### 2.2.6.3 Caveats and Advice for Routing Queues

- Routing loops should be avoided. If a job makes more than 20 routing hops, it is discarded, and mail may be sent. Avoid setting a routing queue’s destination to be the routing queue itself.
- When routing to a complex that is using failover, it’s a good idea to include the names of both primary and secondary servers in a routing destination:
  
  ```
  route_destinations = "destQ@primary_server, destQ@secondary_server"
  ```

- When routing a job between complexes, the job’s owner must be able to submit a job to the destination complex.
- When routing to a destination in another complex, the source and destination complexes should use the same version of PBS. If not, you may need a submission hook to modify incoming jobs.
- It is recommended to list the destination queues in order of the most restrictive first, because the first queue which meets the job’s requirements and is enabled will be its destination.

### 2.2.6.4 Using Resources to Route Jobs Between Queues

You can use resources to direct jobs to the desired queues. The server will automatically route jobs that are in routing queues, based on job resource requests. The destination queue can be at the local server or at another server. If you have more than one PBS complex, you may want to route jobs between the complexes, depending on the resources available at each complex.
You can set up queues for specific kinds of jobs, for example jobs requesting very little memory, a lot of memory, or a particular application. You can then route jobs to the appropriate queues.

A routing queue tests destination queues in the order listed in the queue’s route_destinations attribute. The job is placed in the first queue that meets the job’s request and is enabled.

Please read all of the subsections for this section.

2.2.6.4.i How Queue and Server Limits Are Applied, Except Running Time

The following applies to all resources except for min_walltime and max_walltime.

You can set a minimum and a maximum for each resource at each queue using the resources_min.<resource> and resources_max.<resource> queue attributes. Any time a job is considered for entry into a queue, the job’s resource request is tested against resources_min.<resource> and resources_max.<resource> for that queue. The job’s resource request must be greater than or equal to the value specified in resources_min.<resource>, and less than or equal to the value specified in resources_max.<resource>.

The job is tested only against existing resources_min.<resource> and resources_max.<resource> for the queue.

Only those resources that are specified in the job’s resource request are tested, so if a job does not request a particular resource, and did not inherit a default for that resource, the minimum and maximum tests for that resource are not applied to the job.

If you want jobs requesting only a specific value for a resource to be allowed into a queue, set the queue’s resources_min.<resource> and resources_max.<resource> to the same value. This resource can be numeric, string, string array, or Boolean.

If you limit queue access using a string array, a job must request one of the values in the string array to be allowed into the queue. For example, if you set resources_min.strarr and resources_max.strarr to “blue,red,black”, jobs can request –l strarr=blue, -l strarr=red, or –l strarr=black to be allowed into the queue.

2.2.6.4.ii How Queue and Server Running Time Limits are Applied

For shrink-to-fit jobs, running time limits are applied to max_walltime and min_walltime, not walltime. To set a running time limit for shrink-to-fit jobs, you cannot use resources_max or resources_min for max_walltime or min_walltime. Instead, use resources_max.walltime and resources_min.walltime. See section 4.8.41.6, “Shrink-to-fit Jobs and Resource Limits”, on page 267.
You can route jobs using the following kinds of resources:

- Any server-level or queue-level (job-wide) built-in or custom resource, whether it is numeric, string, or Boolean, for example ncpus and software

  When routing jobs with min_walltime and/or max_walltime, PBS examines the values for resources_min.walltime and resources_max.walltime at the server or queue. See section 2.2.6.4.ii, “How Queue and Server Running Time Limits are Applied”, on page 26.

- The following built-in chunk-level resources:
  - accelerator_memory
  - mem
  - mpiprocs
  - naccelerators
  - ncpus
  - netwins
  - nodect
  - vmem

- Custom vnode-level (chunk-level) resources that are global and have the n, q, or f flags set

- Any resource in the job’s Resource_List attribute; see section 5.9.2, “Resources Requested by Job”, on page 305. For string or string array resources, see section 2.2.6.4.iv, “Using String, String Array, and Boolean Values for Routing and Admittance”, on page 27.

When jobs are routed using a chunk-level resource, routing is based on the sum of that resource across all chunks.

**2.2.6.4.iv** Using String, String Array, and Boolean Values for Routing and Admittance

When using strings or string arrays for routing or admittance, you can use only job-wide (server-level or queue-level) string or string array resources. String or string array resources in chunks are ignored. The resources_min and resources_max attributes work as expected.
Chapter 2 Configuring the Server and Queues

with numeric values. In addition, they can be used with string and Boolean values to force an exact match; this is done by setting both to the same value. For example, to limit jobs entering queue big to those that specify arch=unicos8, or that do not specify a value for arch:

Qmgr: set q ApplQueue resources_max.software=App1
Qmgr: set q ApplQueue resources_min.software=App1

2.2.6.4.v Examples of Routing Jobs

You can force all jobs into a routing queue, or you can allow users to request some queues but not others. If you set up the default queue be a routing queue, and make all execution queues accept jobs only from routing queues, all jobs are initially forced into a routing queue.

Alternatively, you can set up one routing queue and a couple of execution queues which accept jobs only from routing queues, but add other queues which can be requested. Or you could allow jobs to request the execution queues, by making the execution queues also accept jobs that aren’t from routing queues.

Example 2-1: Jobs can request one execution queue named WorkQ. All jobs that do not request a specific queue are routed according to their walltime:

• Create a routing queue RouteQ and make it the default queue:
  Qmgr: create queue RouteQ queue_type = route
  Qmgr: set server default_queue = RouteQ

• Create two execution queues, LongQ and ShortQ. One is for long-running jobs, and one
is for short-running jobs:

```bash
Qmgr: create queue LongQ queue_type = execution
Qmgr: create queue ShortQ queue_type = execution
```

- Set `resources_min.walltime` and `resources_max.walltime` on these queues:
  ```bash
  Qmgr: set queue LongQ resources_min.walltime = 5:00:00
  Qmgr: set queue ShortQ resources_max.walltime = 4:59:00
  ```

- For `LongQ` and `ShortQ`, disallow jobs that are not from a route queue:
  ```bash
  Qmgr: set queue LongQ from_route_only = True
  Qmgr: set queue ShortQ from_route_only = True
  ```

- Set the destinations for `RouteQ` to be `LongQ` and `ShortQ`:
  ```bash
  Qmgr: set queue RouteQ route_destinations = "ShortQ, LongQ"
  ```

- Create a work queue that can be requested:
  ```bash
  Qmgr: create queue WorkQ queue_type = execution
  ```

- Enable and start all queues:
  ```bash
  Qmgr: active queue RouteQ,LongQ,ShortQ,WorkQ
  Qmgr: set queue enabled = True
  Qmgr: set queue started = True
  ```

- Set default for `walltime` at the server so that jobs that don’t request it inherit the default, and land in `ShortQ`:
  ```bash
  Qmgr: set server resources_default.walltime = 4:00:00
  ```

Example 2-2: Jobs are not allowed to request any queues. All jobs are routed to one of three queues based on the job’s `walltime` request:

- Create a routing queue `RouteQ` and make it the default queue:
  ```bash
  Qmgr: create queue RouteQ queue_type = route
  Qmgr: set server default_queue = RouteQ
  ```

- Create three execution queues, `LongQ`, `MedQ`, and `ShortQ`. One is for long-running
Chapter 2  Configuring the Server and Queues

jobs, one is for medium jobs, and one is for short-running jobs:

Qmgr: create queue LongQ queue_type = execution
Qmgr: create queue MedQ queue_type = execution
Qmgr: create queue ShortQ queue_type = execution

• Set resources_min.walltime and resources_max.walltime on these queues:

Qmgr: set queue LongQ resources_min.walltime = 10:00:00
Qmgr: set queue MedQ resources_max.walltime = 9:59:00
Qmgr: set queue MedQ resources_min.walltime = 5:00:00
Qmgr: set queue ShortQ resources_max.walltime = 4:59:00

• For LongQ, MedQ, and ShortQ, disallow jobs that are not from a route queue:

Qmgr: set queue LongQ from_route_only = True
Qmgr: set queue MedQ from_route_only = True
Qmgr: set queue ShortQ from_route_only = True

• Set the destinations for RouteQ to be LongQ, MedQ and ShortQ:

Qmgr: set queue RouteQ route_destinations = “ShortQ, MedQ, LongQ”

• Enable and start all queues:

Qmgr: active queue RouteQ,LongQ,ShortQ,MedQ
Qmgr: set queue enabled = True
Qmgr: set queue started = True

2.2.6.4.vi Caveats for Queue Resource Limits

If a job is submitted without a request for a particular resource, and no defaults for that resource are set at the server or queue, and either the server or queue has resources_max.<resource> set, the job inherits that maximum value. If the queue has resources_max.<resource> set, the job inherits the queue value, and if not, the job inherits the server value.

2.2.6.5 Using Access Control to Route Jobs

You can route jobs based on job ownership by setting access control limits at destination queues. A queue’s access control limits specify which users or groups are allowed to have jobs in that queue. Default behavior is to disallow an entity that is not listed, so you need only list allowed entities.

To set the list of allowed users at a queue:

Qmgr: set queue <queue name> acl_users = “User1@*.example.com, User2@*.example.com”
To enable user access control at a queue:

```
Qmgr: set queue <queue name> acl_user_enable = True
```

To set the list of allowed groups at a queue:

```
Qmgr: set queue <queue name> acl_groups = "Group1@*.example.com,
Group2@*.example.com"
```

To enable group access control at a queue:

```
Qmgr: set queue <queue name> acl_group_enable = True
```

For a complete explanation of access control, see section 8.3, “Using Access Control”, on page 647.

### 2.2.6.6 Allowing Routing of Held or Waiting Jobs

By default, PBS will not route jobs that are held. You can allow a routing queue to route held jobs by setting the queue’s `route_held_jobs` attribute to `True`:

```
Qmgr: set queue <queue name> route_held_jobs = True
```

By default, PBS will not route jobs whose `execution_time` attribute has a value in the future. You can allow a routing queue to route jobs whose start time is in the future by setting the queue’s `route_waiting_jobs` attribute to `True`:

```
Qmgr: set queue <queue name> route_waiting_jobs = True
```

### 2.2.6.7 Setting Routing Retry Time

The default time between routing retries is 30 seconds. To set the time between routing retries, set the value of the queue’s `route_retry_time` attribute:

```
Qmgr: set queue <queue name> route_retry_time = <value>
```

### 2.2.6.8 Specifying Job Lifetime in Routing Queue

By default, PBS allows a job to exist in a routing queue for an infinite amount of time. To change this, set the queue’s `route_lifetime` attribute:

```
Qmgr: set queue <queue name> route_lifetime = <value>
```

### 2.2.7 Queue Requirements

- Each queue must have a unique name. The name must be alphanumeric, and must begin
A Server may have multiple queues of either or both types, but there must be at least one execution queue defined.

2.2.8 Queue Configuration Attributes

Queue configuration attributes fall into three groups:

- Those which apply to both types of queues
- Those which apply only to execution queues
- Those which apply only to routing queues

If an “execution queue only” attribute is set for a routing queue, or vice versa, it is ignored. However, as this situation might indicate the administrator made a mistake, the Server will write a warning message on stderr about the conflict. The same message is written when the queue type is changed and there are attributes that do not apply to the new type.

See “Queue Attributes” on page 354 of the PBS Professional Reference Guide for a table of queue attributes.

2.2.9 Viewing Queue Status

To see the status of a queue, including values for attributes, use the qstat command:

```
qstat -Qf <queue name>
```

To see the status of all queues:

```
qstat -Qf
```

The status of the queue is reported in the State field. The field shows two letters. One is either E (enabled) or D (disabled.) The other is R (running, same as started) or S (stopped.) Attributes with non-default values are displayed. See “qstat” on page 196 of the PBS Professional Reference Guide.

The following queue attributes contain queue status information:
total_jobs
state_count
resourcesAssigned
hasnodes
enabled
started

### 2.2.10 Deleting Queues

Use the `qmgr` command to delete queues.

```
Qmgr: delete queue <queue name>
```

#### 2.2.10.1 Caveats for Deleting Queues

- A queue that has queued or running jobs cannot be deleted.
- A queue that is associated with a vnode via that vnode’s `queue` attribute cannot be deleted. To remove the association, save the output of `pbsnodes -a` to a file and search for the queue. Unset the `queue` attribute for each associated vnode.

### 2.2.11 Defining Queue Resources

For each queue, you can define the resources you want to have available at that queue. To set the value for an existing resource, use the `qmgr` command:

```
Qmgr: set queue <queue name> resourcesAvailable.<resource name> = <value>
```

For example, to set the value of the Boolean resource `RunsMyApp` to `True` at `QueueA`:

```
Qmgr: set queue QueueA resourcesAvailable.RunsMyApp = True
```

For information on how to define a new resource at a queue, see section 5.14, “Custom Resources”, on page 319.

For information on defining default resources at a queue, see section 5.9.3.3, “Specifying Job-wide Default Resources at Queue”, on page 307 and section 5.9.3.4, “Specifying Chunk Default Resources at Queue”, on page 307.
2.2.12 Setting Queue Resource Defaults

The jobs that are placed in a queue inherit the queue’s defaults for any resources not specified by the job’s resource request. You can specify each default resource for each queue. This is described in section 5.9.3, “Specifying Job Default Resources”, on page 305. Jobs inherit default resources according to the rules described in section 5.9.4, “Allocating Default Resources to Jobs”, on page 309.

2.2.13 How Default Server and Queue Resources Are Applied When Jobs Move

When a job is moved from one server to another, the following changes happen:

• Any default resources that were applied by the first server are removed
• Default resources from the new server are applied to the job

When a job is moved from one queue to another, the following changes happen:

• Any default resources that were applied by the first queue are removed
• Default resources from the new queue are applied to the job

For more details on how default resources are inherited when a job is moved, see section 5.9.4.3, “Moving Jobs Between Queues or Servers Changes Defaults”, on page 310.

2.2.14 Specifying Default Queue

PBS has a default execution queue named workq, where jobs are placed when no queue is requested. You can specify which queue should be the default. To specify the queue which is to accept jobs when no queue is requested, set the server’s default_queue attribute to the name of the queue:

```
Qmgr: set server default_queue = <queue name>
```

2.2.15 Associating Queues and Vnodes

You can set up vnodes so that they accept jobs only from specific queues. See section 4.8.2, “Associating Vnodes with Queues”, on page 121.

2.2.16 Configuring Access to Queues

You can configure each queue so that only specific users or groups can submit jobs to the queue. See section 8.3, “Using Access Control”, on page 647.
2.2.17 Setting Limits on Usage at Queues

You can set limits on different kinds of usage at each queue:

- You can limit the size of a job array using the `max_array_size` queue attribute.
- You can limit the number of jobs or the usage of each resource by each user or group, or overall. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367.

2.2.18 Additional Queue Information

For a description of each queue attribute, see “Queue Attributes” on page 354 of the PBS Professional Reference Guide.

For information on using queues for scheduling, see section 4.5, “Using Queues in Scheduling”, on page 114.
Chapter 2  Configuring the Server and Queues
Chapter 3

Configuring MoMs and Vnodes

The installation process creates a basic MoM and vnode configuration which contains the minimum necessary in order to run PBS jobs. This chapter describes how to customize your MoM and vnode configuration.

3.1 Vnodes: Virtual Nodes

A virtual node, or vnode, is an abstract object representing a set of resources which form a usable part of a machine. This could be an entire host, or a nodeboard or a blade. A single host can be made up of multiple vnodes. Each vnode can be managed and scheduled independently. PBS views hosts as being composed of one or more vnodes.

Each vnode has an associated set of attributes and resources. Vnode attributes are listed and described in “Vnode Attributes” on page 366 of the PBS Professional Reference Guide. Vnode resources can be built-in or custom (defined by you.) See Chapter 5, "PBS Resources", on page 287. Rules for setting values for attributes and resources are given in section 3.5.2, “Choosing Configuration Method”, on page 52.

3.1.1 Vnode State

The state of each vnode is controlled by its state attribute. The state of the vnode publishes whether the vnode can accept new jobs, what it is doing, and whether it is usable. The state attribute can take zero or more of the values listed in “Vnode States” on page 412 of the PBS Professional Reference Guide. The state of a vnode can be set by PBS or in a hook. A vnode’s state can be set to offline using the qmgr command; no other values can be set using qmgr.
### 3.1.2 Relationship Between Hosts, Nodes, and Vnodes

A host is any computer. Execution hosts used to be called nodes. However, some machines such as the Altix can be treated as if they are made up of separate pieces containing CPUs, memory, or both. Each piece is called a vnode. See "Vnodes: Virtual Nodes" on page 37. Some hosts have a single vnode and some have multiple vnodes. PBS treats all vnodes alike in most respects.

### 3.1.3 Natural Vnodes

For machines that have more than one vnode, there is a vnode called the natural vnode. A natural vnode does not correspond to any actual hardware. The natural vnode is used to define any placement set information that is invariant for a given host. See section 4.8.32, “Placement Sets”, on page 211. The natural vnode is also used to define dynamic host-level resources, and can be used to define shared resources. On a multi-vnoded machine which has a natural vnode, anything set in the mom_resources line in PBS_HOME/sched_priv/sched_config is shared by all of that machine’s vnodes. See section 5.14.5.1, “Dynamic Host-level Resources”, on page 337 and section 5.4.7, “Shared and Non-shared Vnode Resources”, on page 296.

### 3.1.4 Breaking Chunks Across Vnodes

Chunks can be broken up across vnodes that are on the same host. This is generally used for jobs requesting a single chunk. On the Altix, the scheduler will share memory from a chunk even if all the CPUs are used by other jobs. It will first try to put a chunk entirely on one vnode. If it can, it will run it there. If not, it will break the chunk up across any vnode it can get resources from, even for small amounts of unused memory.

#### 3.1.4.1 Restrictions on Natural Vnode on cpuset Machines

- On a machine that has cpusets, the natural vnode should not have its schedulable resources (ncpus, mem, vmem) set. Leave these resources unset. If these are set by the administrator, their values are retained across restarts until they are changed again or until the vnode is re-created. Setting the values via qmgr will lead the Server and the MoM to disagree on the values.

- On the natural vnode, all values of resources_available.<resource> should be zero (0), unless the resource is being shared among other vnodes via indirection.
3.1.5 Creating Vnodes

3.1.5.1 Creating Vnodes on Single-vnode Machines

For a machine which will have a single vnode:
1. Start MoM on the host where you will create the vnode.
2. Get the short name returned by the `gethostname` command where you will run the MoM.
3. Use the `qmgr` command to create the vnode. Use the name returned by `gethostname`:
   \[ \text{Qmgr: create node <vnode name> [<attribute>=<value>]} \]
   Attributes and their possible values are listed in “Vnode Attributes” on page 366 of the PBS Professional Reference Guide.
   All comma-separated attribute-value strings must be enclosed in quotes.

3.1.5.2 Creating Vnodes on Multi-vnode Machines

For a machine which will have more than one vnode, you create the natural vnode, but PBS handles creation of the other vnodes:
1. For machines such as an Altix, you must start PBS on the multi-vnode host using the PBS start/stop script. See “The PBS Start/Stop Script” on page 177 in the PBS Professional Installation & Upgrade Guide.
2. Get the short name returned by the `gethostname` command where you will run the MoM.
3. Use the `qmgr` command to create the natural vnode. Use the name returned by `gethostname`:
   \[ \text{Qmgr: create node <natural vnode name> [<attribute>=<value>]} \]
   Attributes and their possible values are listed in “Vnode Attributes” on page 366 of the PBS Professional Reference Guide.
   All comma-separated attribute-value strings must be enclosed in quotes.

After you create the natural vnode, the other vnodes become available for use. Follow the rules for configuring these machines in section 3.5.2, “Choosing Configuration Method”, on page 52. See section 10.4, “Support for SGI”, on page 808.
Chapter 3  Configuring MoMs and Vnodes

Here is an example of the vnode definition for a natural vnode on an Altix:

```bash
altix03: pnames = cbrick, router
altix03: sharing = ignore_excl
altix03: resources_available.ncpus = 0
altix03: resources_available.mem = 0
altix03: resources_available.vmem = 0
```

For machines such as a Cray, creation of vnodes other than the natural vnode is handled by MoM. You create the natural vnode using `qmgr`:

```bash
Qmgr: create node <natural vnode name>
```


3.1.5.2.i  Caveats for Creating Vnodes

- On the Cray, when creating a vnode to represent a login node, use the short name returned by the `gethostname` command on the login node. For example, if `gethostname` returns `HostA`, do the following:

  ```bash
  Qmgr: create node HostA
  ```

  If you create a vnode with a different name from the short name returned by `gethostname`, the following happens:

  - MoM creates a vnode whose name is the short name returned by `gethostname`
  - The vnode you created is not recognized by MoM, and is marked stale

- It is not a good idea to try to use `qmgr` to create the vnodes for an Altix or UV, other than the natural vnode. You do need to create the natural vnode via `qmgr`. It is possible to use `qmgr` to create a vnode with any name. The "[x]" naming does not imply any special significance; it just an internal convention for naming vnodes on an Altix or UV. The fact that you can create a vnode with a weird name does not mean however that the MoM on the host knows about that vnode. If the MoM does not know about the vnode, the vnode will be considered "stale" and not usable. By default, MoM only knows about the natural vnode, the one whose name is the same as the host.

- Vnode attributes cannot be used as vnode names.

3.1.6  Deleting Vnodes

3.1.6.1  Deleting the Vnode on a Single-vnode Machine

Use the `qmgr` command to delete the vnode:

```bash
Qmgr: delete node <vnode name>
```
Optionally, you can stop PBS on the execution host whose vnode was deleted.

### 3.1.6.2 Deleting Vnodes on a Multi-vnode Machine

As long as there is a configuration file describing vnodes, PBS will believe they exist. Therefore, you must first remove the configuration file. To delete one or more vnodes on a multi-vnode machine, follow these steps:

1. Use the `-s remove` option to the `pbs_mom` command to remove the Version 2 configuration file that describes the vnodes to be removed:
   ```
   On UNIX/Linux:
   pbs_mom -s remove <configuration file target>
   On Windows:
   pbs_mom -N -s remove <configuration file target>
   ```

2. Use the `-s insert` option to the `pbs_mom` command to insert a new Version 2 configuration file describing the vnodes to be retained:
   ```
   On UNIX/Linux:
   pbs_mom -s insert <configuration file target> <input file source>
   On Windows:
   pbs_mom -N -s insert <configuration file target> <input file source>
   ```

3. Restart the MoM:
   ```
   <path to start/stop script>/pbs restart
   ```

4. Use the `qmgr` command to remove the vnodes:
   ```
   Qmgr: delete node <vnode name>
   ```

### 3.1.6.3 Deleting Vnodes on a Cray

For information on deleting vnodes on a Cray, see section 10.3.11.6, “Deleting Vnodes on Cray”, on page 802.

### 3.1.7 Allocating Vnodes to Jobs

PBS can run jobs only on the execution hosts that are managed by the PBS server, and running a MoM.
By default, when the scheduler looks for the vnodes on which to run a job, it goes down the list of hosts in the order in which they appear in the server’s list of hosts, and places the job on the first available vnode or vnodes meeting the job’s requirements. This means that the order of the list of hosts affects default job placement. You can specify more sophisticated choices; see Chapter 4, "Scheduling", on page 63.

The scheduler follows the specified rules for selecting vnodes that match each job’s request. Once the scheduler finds the resources that match a job’s request, it allocates vnodes to the job, according to the value of the vnode’s sharing attribute and the job’s resource request.

### 3.1.7.1 Sharing Vnodes Among Jobs

Each vnode can be allocated exclusively to one job, or its resources can be shared among jobs. Hosts can also be allocated exclusively to one job, or shared among jobs.

How vnodes are allocated to jobs is determined by a combination of the vnode’s sharing attribute and the job’s resource request. The possible values for the vnode sharing attribute, and how they interact with a job’s placement request, are described in “sharing” on page 371 of the PBS Professional Reference Guide. A description of how resources are allocated is in section 4.8.40, “Shared vs. Exclusive Use of Resources by Jobs”, on page 261.

If a vnode is allocated exclusively to a job, all of its resources are assigned to the job. The state of the vnode becomes job-exclusive. No other job can use the vnode.

If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either default_exclhost or force_exclhost, all vnodes on that host must have the same value for the sharing attribute. When the MoM starts or restarts, if any vnode on a host is set to either default_exclhost or force_exclhost, and another vnode is set to a different value, the MoM will exit and log the following error message at event class 0x0001:

```
It is erroneous to mix sharing= <sharing val> for vnode <name> with sharing= <force_exclhost|default_exclhost> which is set for other vnodes on host <host>
```

### 3.1.7.2 Placing Jobs on Vnodes

Jobs can be placed on vnodes according to the job’s placement request. Each chunk from a job can be placed on a different host, or a different vnode. Alternatively, all chunks can be taken from a single host, or from chunks sharing the same value for a specified resource. The job can request exclusive use of each vnode, or shared use with other jobs. The job can request exclusive use of its hosts. For details, see "Specifying Job Placement", on page 89 of the PBS Professional User’s Guide.
3.2 MoMs

A MoM daemon runs on each execution host and manages the jobs on that execution host. The `pbs_mom` command starts the PBS job monitoring and execution daemon, called MoM. The `pbs_mom` daemon starts jobs on the execution host, monitors and reports resource usage, enforces resource usage limits, and notifies the server when the job is finished. The MoM also runs any prologue scripts before the job runs, and runs any epilogue scripts after the job runs.

When the MoM starts a job, she creates a new session that is as identical to the user’s login session as is possible. For example, under UNIX, if the user’s login shell is `csh`, then MoM creates a session in which `.login` is run as well as `.cshrc`. MoM returns the job’s output to the user.

The MoM performs any communication with job tasks and with other MoMs. The MoM on the first vnode on which a job is running manages communication with the MoMs on the remaining vnodes on which the job runs. The MoM on the first vnode is called `Mother Superior`.

The MoM log file is in `PBS_HOME/mom_logs`. The MoM writes an error message in its log file when it encounters any error. The MoM also writes other miscellaneous information to its log file. If it cannot write to its log file, it writes to standard error.

The executable for `pbs_mom` is in `PBS_EXEC/sbin`, and can be run only by root.

See “Manually Starting MoM” on page 179 in the PBS Professional Installation & Upgrade Guide for information on starting and stopping MoM.

3.2.1 Single-vnode, Multi-vnode, and Cpusetted Systems

For systems that can be subdivided into more than one virtual node, or vnode, PBS manages each vnode much as if it were a host. On each machine, the MoM manages the vnodes. PBS may treat a host such as an Altix as a set of virtual nodes, in which case one MoM manages all of the host’s vnodes. For details about vnodes, see section 3.1, “Vnodes: Virtual Nodes”, on page 37.

The `pbs_mom` you select to run a machine depends on the type of machine and the way you want it managed. The MoM that manages a system without cpusets is `pbs_mom.standard`. This MoM can manage a single-vnoded or a multi-vnoded, non-cpusetted system. The MoM that has extensions to manage a cpusetted machine such as the Altix is `pbs_mom.cpuset`. The appropriate MoM is copied to `pbs_mom`. See the PBS Professional Installation and Upgrade Guide.
The following sections describe configuration files and methods for all MoMs and vnodes. See section 10.4, "Support for SGI", on page 808 for information that is specific to systems with cpusets.

### 3.3 Files and Directories Used by MoM

If PBS_MOM_HOME is present in the pbs.conf file, pbs_mom will use that directory for its “home” instead of PBS_HOME. Under UNIX/Linux, all files and directories that MoM uses must be owned by root. Under Windows, these directories must have at least Full Control permission for the local Administrators group. MoM uses the following files and directories:

#### UNIX:

**Table 3-1: MoM Files and Directories Under UNIX**

<table>
<thead>
<tr>
<th>File/Directory</th>
<th>Description</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>aux</td>
<td>Directory</td>
<td>0755</td>
</tr>
<tr>
<td>checkpoint</td>
<td>Directory</td>
<td>0700</td>
</tr>
<tr>
<td>checkpoint script</td>
<td>File</td>
<td>0755</td>
</tr>
<tr>
<td>mom_logs</td>
<td>Directory</td>
<td>0755</td>
</tr>
<tr>
<td>mom_priv</td>
<td>Directory</td>
<td>0751</td>
</tr>
<tr>
<td>mom_priv/jobs</td>
<td>Directory</td>
<td>0751</td>
</tr>
<tr>
<td>mom_priv/config</td>
<td>File</td>
<td>0644</td>
</tr>
<tr>
<td>mom_priv/prologue</td>
<td>File</td>
<td>0755</td>
</tr>
<tr>
<td>mom_priv/epilogue</td>
<td>File</td>
<td>0755</td>
</tr>
<tr>
<td>pbs_environment</td>
<td>File</td>
<td>0644</td>
</tr>
<tr>
<td>spool</td>
<td>Directory</td>
<td>1777 (drwxrwxrwt)</td>
</tr>
<tr>
<td>undelivered</td>
<td>Directory</td>
<td>1777 (drwxrwxrwt)</td>
</tr>
<tr>
<td>Version 2 configuration files</td>
<td>Files</td>
<td>0755</td>
</tr>
<tr>
<td>PBS reserved configuration files</td>
<td>Files</td>
<td>----</td>
</tr>
</tbody>
</table>
### Configuring MoMs and Vnodes

#### Chapter 3

**Table 3-1: MoM Files and Directories Under UNIX**

<table>
<thead>
<tr>
<th>File/Directory</th>
<th>Description</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job temporary directory</td>
<td>Directory</td>
<td>1777</td>
</tr>
</tbody>
</table>

Windows:

**Table 3-2: MoM Files and Directories Under Windows**

<table>
<thead>
<tr>
<th>File/Directory</th>
<th>Description</th>
<th>Ownership/Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>auxiliary</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group and read-only access to Everyone</td>
</tr>
<tr>
<td>checkpoint</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group</td>
</tr>
<tr>
<td>checkpoint script</td>
<td>File</td>
<td>At least Full Control permission for the local Administrators group</td>
</tr>
<tr>
<td>mom_logs</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group and read-only access to Everyone</td>
</tr>
<tr>
<td>mom_priv</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group and read-only access to Everyone</td>
</tr>
<tr>
<td>mom_priv/jobs</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group and read-only access to Everyone</td>
</tr>
<tr>
<td>mom_priv/config</td>
<td>File</td>
<td>At least Full Control permission for the local Administrators group</td>
</tr>
<tr>
<td>pbs_environment</td>
<td>File</td>
<td>At least Full Control permission for the local Administrators group and read-only to Everyone</td>
</tr>
<tr>
<td>spool</td>
<td>Directory</td>
<td>Full access to Everyone</td>
</tr>
</tbody>
</table>
3.4 MoM and Vnode Configuration Files

The behavior of each MoM is controlled through its configuration files. You configure MoMs by specifying values for parameters in configuration files.

Vnodes are controlled through the values of their attributes. You configure vnodes by specifying values for vnode attributes, either by setting attribute values using the qmgr command or by creating configuration files using the `pbs_mom -s insert` command (`pbs_mom -N -s insert` on Windows).

The method to use to configure MoMs and vnodes depends on the machine being configured. The methods used are described in section 3.5, “How to Configure MoMs and Vnodes”, on page 50.

### 3.4.1 Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

### 3.4.2 Types of MoM and Vnode Configuration Files

MoM and vnode configuration information can be contained in configuration files of three types:

- Version 1
- PBS reserved
- Version 2
3.4.2.1 Version 1 Configuration Files

You edit the Version 1 configuration file directly. The Version 1 configuration file is usually `PBS_HOME/mom_priv/config`. This file contains the parameters that control MoM’s behavior.

The Version 1 configuration file must be secure. It must be owned by a user ID and group ID both less than 10 and must not be world-writable.

For a complete description of the syntax and contents of the Version 1 configuration file, see “MoM Parameters” on page 267 of the PBS Professional Reference Guide.

3.4.2.2 PBS Reserved Configuration Files

PBS reserved configuration files are created by PBS and are prefixed with "PBS". These files are created by PBS and are not configurable. Do not attempt to edit these files. An attempt to create or remove a file with the "PBS" prefix will result in an error.

3.4.2.3 Version 2 Configuration Files

Version 2 configuration files are those created by the site administrator. These files can contain vnode attribute settings. Do not attempt to edit these files directly. Instead of editing these directly, you create a local file and give it as an argument to the `pbs_mom -s insert` option (`pbs_mom -N -s insert` on Windows), and PBS creates a new configuration file for you. See section 3.5.3, “Creating Version 2 MoM Configuration Files”, on page 53. Their syntax is called “Version 2” in order to differentiate it from the syntax of the Version 1 configuration files.

You can list, add, delete and display Version 2 configuration files using the `pbs_mom -s` option (`pbs_mom -N -s` on Windows). See “`pbs_mom`” on page 58 of the PBS Professional Reference Guide for information about `pbs_mom` options.

3.4.2.3.i Removing Version 2 Configuration Files

You can remove a Version 2 configuration file using the `pbs_mom -s remove` option (`pbs_mom -N -s remove` on Windows). See “`pbs_mom`” on page 58 of the PBS Professional Reference Guide.

3.4.3 Location of MoM’s Configuration Files

The Version 1 configuration file is usually `PBS_HOME/mom_priv/config`. It can be in a different location; in that case, MoM must be started with the `-c` option. See “`pbs_mom`” on page 58 of the PBS Professional Reference Guide.
Chapter 3  Configuring MoMs and Vnodes

PBS places PBS reserved and Version 2 configuration files in an area that is private to each installed instance of PBS.

### 3.4.4 Listing and Viewing PBS Reserved and Version 2 Configuration Files

You can list and view the PBS reserved configuration files and the Version 2 configuration files using the `pbs_mom -s list` and `pbs_mom -s show` options (`pbs_mom -N -s list` and `show` on Windows). See “`pbs_mom` on page 58 of the PBS Professional Reference Guide.”

### 3.4.5 Caveats and Restrictions for Configuration Files

- Do not attempt to directly create PBS reserved or Version 2 configuration files; instead, use the `pbs_mom -s` option (`pbs_mom -N -s` on Windows).
- Note that the `-d` option to `pbs_mom` changes where MoM looks for `PBS_HOME`, and using this option will change where MoM looks for all configuration files. If you use the `-d` option, MoM will look in the new location for any PBS reserved and Version 2 files.
- The `-c` option will change which Version 1 configuration file MoM reads.
- Do not move PBS reserved configuration files.
- If you set a value using `qmgr`, this value overrides the value specified in a configuration file.
- Do not mix the configuration file contents or syntax. Each type must use its own syntax, and contain its own type of information.
- When you create a Version 2 configuration file for a pre-existing vnode, make sure it specifies all of the information about the vnode, such as resources and attribute settings. The creation of the configuration file overrides previous settings, and if the new file contains no specification for a resource or attribute, that resource or attribute becomes unset.
- Version 2 configuration files can be moved from one installed instance of PBS to another. To move a set of Version 2 configuration files from one installed instance of PBS to another:
  1. Use the `-s list` directive with the "source" instance of PBS to enumerate the Version 2 files.
  2. Use the `-s show` directive with each Version 2 file of the "source" instance of PBS to save a copy of that file.
  3. Use the `-s insert` directive with each file at the "target" instance of PBS to create a copy of each Version 2 configuration file.
3.4.5.1 When MoM Reads Configuration Files

MoM reads the configuration files at startup and reinitialization. On UNIX, this is when `pbs_mom` receives a SIGHUP signal or is started or restarted, and on Windows, when MoM is started or restarted. In order for any configuration changes to take effect, MoM must be HUPed.

If you make changes to the hardware or a change occurs in the number of CPUs or amount of memory that is available to PBS, such as a non-PBS process releasing a cpuset, you should restart PBS, by typing the following:

```
<path-to-script>/pbs restart
```

The MoM daemon is normally started by the PBS start/stop script.

When MoM is started, it will open its Version 1 configuration file, `mom_priv/config`, in the path specified in `pbs.conf`, if the file exists. If it does not, MoM will continue anyway. The `config` file may be placed elsewhere or given a different name, by starting `pbs_mom` using the `-c` option with the new file and path specified. See “Manually Starting MoM” on page 179 in the PBS Professional Installation & Upgrade Guide.

The files are processed in this order:

1. Version 1 configuration file
2. PBS reserved configuration files
3. Version 2 configuration files

Within each category, the files are processed in lexicographic order.

The contents of a file that is read later will override the contents of a file that is read earlier.
Chapter 3  Configuring MoMs and Vnodes

3.5 How to Configure MoMs and Vnodes

3.5.1 Configuration Methods

The method you use to configure MoMs and vnodes depends upon the machine being configured. The methods are the following:

<table>
<thead>
<tr>
<th>Method</th>
<th>When Method Changes MoM Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the <code>qmgr</code> command to set attribute values</td>
<td>Immediately</td>
</tr>
<tr>
<td>Editing the Version 1 configuration file PBS_HOME/mom_priv/config</td>
<td>When MoM is restarted</td>
</tr>
<tr>
<td>Using the <code>pbs_mom -s insert</code> command to create a configuration file (<code>pbs_mom -N -s insert</code> on Windows)</td>
<td>When MoM is restarted</td>
</tr>
<tr>
<td>Using the <code>pbsnodes</code> command to change the state of a vnode</td>
<td>Immediately</td>
</tr>
</tbody>
</table>

3.5.1.1 The `qmgr` Command

You use the `qmgr` command to set attribute values. You can use the `qmgr` command to set attribute values for individual vnodes where those vnodes are part of a multi-vnode machine.

To set a vnode’s attribute, the format is the following:

```
qmgr -c 'set node <vnode name> <attribute> = <value>'
```

or start `qmgr`, and use the following:

```
set node <vnode name> <attribute> = <value>
```

The `qmgr` command is described in “`qmgr` on page 150 of the PBS Professional Reference Guide.”

If you set a value using `qmgr`, this value overrides the value specified in a configuration file.

3.5.1.2 Editing Version 1 Files

Use your favorite text editor to edit Version 1 configuration files.
When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

3.5.1.3 Using the `pbs_mom -s insert` Command

You use the `pbs_mom -s insert` command (`=N -s insert` on Windows) to create all Version 2 configuration files. First, you create a script which is to be the contents of the configuration file. Then, you insert the script using the `pbs_mom -s insert` command:

UNIX/Linux:

```
pbs_mom -s insert <script> <configuration file name>
```

Windows:

```
pbs_mom -N -s insert <script> <configuration file name>
```

For a description of the Version 2 syntax, see section 3.5.3, “Creating Version 2 MoM Configuration Files”, on page 53.

3.5.1.4 Using the `pbsnodes` Command

The `pbsnodes` command is used to set the state of a host to be `offline` or not `offline`. To set the state attribute of one or more hosts to `offline`:

```
pbsnodes -o <hostname [hostname ...]>
```

To remove the `offline` setting from the state attribute of one or more hosts:

```
pbsnodes -r <hostname [hostname ...]>
```

Note that the `pbsnodes` command operates on hosts, not individual vnodes where those vnodes are on multi-vnode machines. To operate on individual vnodes, use the `qmgr` command.

See “`pbsnodes`” on page 103 of the PBS Professional Reference Guide.
3.5.2 Configuring MoMs and Vnodes

3.5.2.1 Configuring Single-vnode Machines without cpusets

To configure the MoM and vnode on a single-vnode machine without cpusets, do the following:

- To configure MoM, including local resources, edit the Version 1 MoM parameter file
- To configure vnodes, use the qmgr command to set vnode attributes and global resources

3.5.2.1.i Exceptions

- Use `pbs_mom -s insert (pbs_mom -N -s insert on Windows)` to set the sharing vnode attribute

3.5.2.2 Configuring Multi-vnode Machines without cpusets

To configure the MoM and vnodes on a multi-vnode machine without cpusets, do the following:

- To configure MoM, including local resources, edit the Version 1 MoM parameter file
- To configure vnodes, use the qmgr command to set vnode attributes and global resources

3.5.2.2.i Exceptions

- Use the `pbs_mom -s insert` command (`pbs_mom -N -s insert on Windows`) to set the sharing vnode attribute (vnode definition files are not recommended on Cray)
- You can use `pbsnodes` to set the state vnode attribute

3.5.2.2.ii Restrictions

- Set the Mom vnode attribute for the natural vnode only.

3.5.2.3 Configuring Machines with Cpusets

To configure the MoM and vnodes on a machine that has cpusets, do the following:

- To configure MoM, including local resources, edit the Version 1 MoM parameter file
- To configure vnodes, use the `pbs_mom -s insert` command (`pbs_mom -N -s insert on Windows`) to set vnode attributes and global resources.
3.5.2.3.i Exceptions

- You can use `qmgr` or `pbsnodes` to set the `state` vnode attribute
- Use `qmgr` to set the `priority` vnode attribute

3.5.2.3.ii Restrictions

- Do not use `qmgr` to configure vnodes, especially for sharing, `resources_available.ncpus`, `resources_available.vmem`, and `resources_available.mem`.
- Do not attempt to set values for `resources_available.ncpus`, `resources_available.vmem`, or `resources_available.mem`. These are set by PBS when the topology file is read.
- Set the `Mom` vnode attribute for the natural vnode only. Do not attempt to set it for any other vnodes.

3.5.3 Creating Version 2 MoM Configuration Files

3.5.3.1 Operating on Version 2 Configuration Files

You can list, add, delete and display Version 2 configuration files using the `pbs_mom -s` option (`pbs_mom -N -s` on Windows). See “`pbs_mom`” on page 58 of the PBS Professional Reference Guide for information about `pbs_mom` options.

3.5.3.2 Format of Version 2 Configuration Files

Any Version 2 configuration file must begin with this line:

```
$configversion 2
```

The format of the remaining contents of the file is the following:

```
<vnode ID> : <attribute name> = <attribute value>
```

where

<table>
<thead>
<tr>
<th>Table 3-4: Elements in Version 2 Reserved Configuration Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td><code>&lt;vnode ID&gt;</code></td>
</tr>
<tr>
<td><code>&lt;attribute name&gt;</code></td>
</tr>
</tbody>
</table>
A vnode's ID is an identifier that will be unique across all vnodes known to a given `pbs_server` and will be stable across reinitializations or invocations of `pbs_mom`. ID stability is important when a vnode's CPUs or memory might change over time and PBS is expected to adapt to such changes by resuming suspended jobs on the same vnodes to which they were originally assigned. Vnodes for which this is not a consideration may simply use IDs of the form "0", "1", etc. concatenated with some identifier that ensures uniqueness across the vnodes served by the `pbs_server`. Vnode attributes cannot be used as vnode names. See “Vnode Attributes” on page 366 of the PBS Professional Reference Guide, where vnode attributes are listed.

### 3.5.3.3 Using the `pbs_mom -s insert` Command

To create a Version 2 configuration file:

1. Create the script that is to be the contents of the configuration file
2. Make this script into a configuration file using the `pbs_mom -s insert` command.

Example 3-1: If your machine has 4 vnodes, named `BigNode0`, `BigNode1`, `SmallNode0`, and `SmallNode1`, and you want big jobs to have exclusive use of their vnodes, and small...
jobs to share their vnodes, then set sharing for big and small vnodes by creating a file "set_sharing" containing the following:

```
$configversion 2
BigNode0: sharing = default_excl
BigNode1: sharing = default_excl
SmallNode0: sharing = default_shared
SmallNode1: sharing = default_shared
```

Then use the `pbs_mom -s insert <filename> <script>` option to create the configuration file:

UNIX/Linux:

```
pbs_mom -s insert sharing_config set_sharing
```

Windows:

```
pbs_mom -N -s insert sharing_config set_sharing
```

The script `sharing_config` is the new Version 2 configuration file. Its contents will override previously-read sharing settings. You must restart the MoM after changing the configuration file.

Example 3-2: To change the sharing attribute on the host named `host3`:

1. Check that `pbsnodes` shows host3 has “sharing = default_shared”:
   ```
   pbsnodes host3
   ```

2. Change the setting to be “sharing = force_excl”:
   ```
   As root, create a script file /tmp/excl_file which contains the following:
   `$configversion 2
   <host3>: sharing=force_excl
   `)
   ```

3. With the `pbs_mom` daemon running, execute the following on host3:
   ```UNIX/Linux:
   # $PBS_EXEC/sbin/pbs_mom -s insert excl /tmp/excl_file
   ```

4. Check that this took effect. The following should show "excl".
   ```UNIX/Linux:
   # $PBS_EXEC/sbin/pbs_mom -s list
   ```

5. Restart `pbs_mom`:
   ```
   # kill -HUP <PID of pbs_mom>
   ```

6. Check that “pbsnodes host3” now shows “sharing = force_excl”
3.5.3.4 Caveats and Restrictions for `pbs_mom -s insert`

On Windows, the `pbs_mom -s` option must be used with the `-N` option so that MoM will start in standalone mode.

3.5.4 Using `qmgr` to Set Vnode Resources and Attributes

One of the PBS reserved configuration files is `PBSvnodedefs`, which is created by a placement set generation script. You can use the output of the placement set generation script to produce input to `qmgr`. The placement set generation script normally emits data for the `PBSvnodedefs` file. If the script is given an additional “-v type=q” argument it emits data in a form suitable for input to `qmgr`:

```
set node <ID> resources_available.<ATTRNAME> = <ATTRVALUE>
```

where `<ID>` is a vnode identifier unique within the set of hosts served by a `pbs_server`. Conventionally, although by no means required, the `<ID>` above will look like `HOST[<localID>]` where `HOST` is the host's FQDN stripped of domain suffixes and `<localID>` is an identifier whose meaning is unique to the execution host on which the referred to vnode resides. For invariant information, it will look like this:

```
set node <ID> priority = 2
```

3.5.5 Caveats and Advice on Configuring MoMs and Vnodes

3.5.5.1 Changing Resource Settings

In general, it is not advisable to set `resources_available.ncpus` or `resources_available.mem` to a value greater than PBS has detected on the machine. This is because you do not want MoM to try to allocate more resources than are available.

In general, it is safe to set `resources_available.ncpus` or `resources_available.mem` to a value less than PBS has detected.

3.5.5.2 Resource Values for Natural Vnode

On the natural vnode, all values for `resources_available.<resource>` should be zero (0), unless the resource is being shared among other vnodes via indirection.
3.5.5.3 Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

3.6 Configuring MoM and Vnode Features

3.6.1 Configuring MoM’s Polling Cycle

3.6.1.1 Polling on UNIX/Linux

In this section, we describe how to configure MoM’s polling cycle. Please note that polling intervals cannot be considered to be exact:

- The calculation below simply provides a minimum amount of time between one poll and the next.
- The actual time between polls can vary. The actual time taken by MoM also depends on the other tasks MoM is performing, such as starting jobs, running a prologue or epilogue, etc.
- The timing of MoM’s activities is not completely under her control, because she is a user process.
- The finest granularity for calculating polling is in seconds.

MoM’s polling cycle is determined by the values of $\text{min\_check\_poll}$ and $\text{max\_check\_poll}$. The interval between each poll starts at $\text{min\_check\_poll}$ and increases with each cycle until it reaches $\text{max\_check\_poll}$, after which it remains the same. The amount by which the cycle increases is the following:

\[
\frac{( \text{max\_check\_poll} - \text{min\_check\_poll} + 19 )}{20}
\]

The default value for $\text{max\_check\_poll}$ is 120 seconds. The minimum is 1 second. It is not recommended to set $\text{max\_check\_poll}$ to less than 30 seconds.

The default value for $\text{min\_check\_poll}$ is 10 seconds. The minimum is 1 second. It is not recommended to set $\text{min\_check\_poll}$ to less than 10 seconds.

The start of a new job resets the polling for all of the jobs being managed by this MoM.

Chapter 3  Configuring MoMs and Vnodes

3.6.1.2  Polling on Windows

On Windows, MoM updates job usage at fixed intervals of 10 seconds. The $min_check_poll and $max_check_poll parameters are not used by MoM on Windows. MoM looks for any job that has exceeded a limit for *walltime*, *mem*, or *cput*, and terminates jobs that have exceeded the limit.

3.6.1.2.i  Windows Polling Caveats

The *ncpus* resource cannot be tracked in Windows.

3.6.1.3  How Polling is Used

- Job-wide limits are enforced by MoM using polling. See section 5.15.3.4.i, “Job Memory Limit Enforcement on UNIX”, on page 395. MoM can enforce *cpuaverage* and *cpuburst* resource usage. See section 5.15.3.5.i, “Average CPU Usage Enforcement”, on page 397 and section 5.15.3.5.ii, “CPU Burst Usage Enforcement”, on page 398.

- MoM enforces the $restrict_user access restrictions on the polling cycle controlled by $min_check_poll and $max_check_poll. See section 3.6.6, “Restricting User Access to Execution Hosts”, on page 59.

- Cycle harvesting has its own polling interval. See “$kbd_idle <idle_wait> <min_use> <poll_interval>” on page 273 of the PBS Professional Reference Guide for information on $kbd_idle.

3.6.1.4  Recommendations for Polling Interval

Do not set $max_check_poll to less than 30 seconds.

Do not set $min_check_poll to less than 10 seconds.

If you have many small jobs, frequent polling can take up a lot of MoM’s cycles. You may want to set $min_check_poll and $max_check_poll to somewhat higher values.

3.6.2  Configuring Host-level Resources

Before configuring host-level resources, consider how you will use them. When configuring static resources, it is best to configure global static resources. Even though they are global, they can be configured at the host level. Global resources can be operated on via the qmgr command and viewed via the qstat command. When configuring dynamic resources, if you need the script to run at the execution host, configure local dynamic resources. These resources cannot be operated on via the qmgr command or viewed via the qstat command.
3.6.2.1 Configuring Global Static Vnode Resources

You can create global custom static host-level resources that can be reported by MoM and used for jobs. Follow the instructions in section 5.14.5.2, “Static Host-level Resources”, on page 339.

You can set values for built-in and custom global static vnode resources according to the rules in section 3.5.2, “Choosing Configuration Method”, on page 52.

3.6.2.1.i Configuring Local Dynamic Vnode Resources

You can create local custom dynamic host-level resources. The primary use of this feature is to add site-specific resources, such as software application licenses or scratch space. Follow the instructions in section 5.14.5.1, “Dynamic Host-level Resources”, on page 337.

3.6.3 Manual Creation of cpusets Not Managed by PBS

You may wish to create cpusets not managed by PBS on an Altix running supported versions of ProPack or SGI Performance Suite. If you have not started PBS, create these cpusets before starting PBS. If you have started PBS, requeue any jobs, stop PBS, create your cpuset(s), then restart PBS.

3.6.4 Configuring Site-Specific Job Termination

For information on site-specific job termination, see section 11.8.4, “Configuring Site-specific Job Termination”, on page 836.

3.6.5 Job Checkpoint and Restart

If you want support for job checkpoint and restart, you can configure MoM to run checkpoint and restart scripts. See section 9.3, “Checkpoint and Restart”, on page 711.

3.6.6 Restricting User Access to Execution Hosts

PBS provides a facility to prevent users who are not running PBS jobs from using machines controlled by PBS. You can turn this feature on by using the $restrict_user MoM directive. This directive can be fine-tuned by using the $restrict_user_exceptions and $restrict_user_maxsysid MoM directives. This feature can be set up host by host.

• A user requesting exclusive access to a set of hosts (via place=excl) can be guaranteed that no other user will be able to use the hosts assigned to his job, and PBS will not
assign any unallocated resources on the vnode to another job.

- A user requesting non-exclusive access to a set of hosts can be guaranteed that no non-PBS users are allowed access to the hosts.
- A privileged user can be allowed access to the complex such that they can log into a host without having a job active.
- An abusive user can be denied access to the complex hosts.

The administrator can find out when users try to access hosts without going through PBS. The administrator can ensure that application performance is consistent on a complex controlled by PBS. PBS will also be able to clean up any job processes remaining after a job finishes running. The log event class for messages concerning restricting users is 0x0002.

For a vnode with access restriction turned on:

- Any user not running a job who logs in or otherwise starts a process on that vnode will have his processes terminated.
- A user who has logged into a vnode where he owns a job will have his login terminated when the job is finished.
- When MoM detects that a user that is not exempt from access restriction is using the system, that user's processes are killed and a log message is output:

```
01/16/2006 22:50:16;0002;pbs_mom;Svr;restrict_user;
    killed uid 1001 pid 13397(bash) with log event class PBSE_SYSTEM.
```

You can set up a list of users who are exempted from the restriction via the `$restrict_user_exceptions` directive. This list can contain up to 10 usernames.

Example 3-3: Turn access restriction on for a given node:
```
$restrict_user on
```

Example 3-4: Limit the users affected to those with a user ID greater than 500:
```
$restrict_user_maxsysid 500
```

Example 3-5: Exempt specific users from the restriction:
```
$restrict_user_exceptions userA, userB, userC
```

Note that a user who has a job running on a particular host will be able to log into that host.

### 3.6.6.1 Windows Restriction

The user access restriction feature is not supported on Windows.
3.6.7 Vnode Resources Set by MoM

If the following vnode resources are not explicitly set, they will take the value provided by MoM. But if they are explicitly set, that setting will be carried forth across Server restarts. They are:

resources_available.ncpus
resources_available.arch
resources_available.mem

3.6.8 Vnode Comments

Vnodes have a comment attribute which can be used to display information about that vnode. If the comment attribute has not been explicitly set by the PBS Manager and the vnode is down, it will be used by the PBS Server to display the reason the vnode was marked down. If the Manager has explicitly set the attribute, the Server will not overwrite the comment. The comment attribute may be set via the qmgr command:

```
Qmgr: set node pluto comment="node will be up at 5pm"
```

Once set, vnode comments can be viewed via pbsnodes, xpbsmon (vnode detail page), and qmgr. See "pbsnodes" on page 103 of the PBS Professional Reference Guide, and "xpbsmon" on page 252 of the PBS Professional Reference Guide.
Chapter 4

Scheduling

The "Scheduling Policy Basics" section of this chapter describes what PBS can do, so that you can consider these capabilities when choosing how to schedule jobs. The "Choosing a Policy" section describes how PBS can meet the scheduling needs of various workloads. The "Scheduling Tools" section describes each scheduling tool offered by PBS.

4.1 Chapter Contents

4.1 Chapter Contents ................................................................. 63
4.2 Scheduling Policy Basics ...................................................... 65
  4.2.1 How Scheduling can be Used ............................................ 65
  4.2.2 What is Scheduling Policy? ............................................. 66
  4.2.3 Basic PBS Scheduling Behavior ....................................... 66
  4.2.4 Sub-goals ........................................................................ 66
  4.2.5 Job Prioritization ............................................................. 67
  4.2.6 Resource Allocation to Users, Projects & Groups ............... 74
  4.2.7 Time Slot Allocation ....................................................... 77
  4.2.8 Job Placement Optimization ........................................... 79
  4.2.9 Resource Efficiency Optimizations ................................. 82
  4.2.10 Overrides ....................................................................... 85
4.3 Choosing a Policy ............................................................... 86
  4.3.1 Overview of Kinds of Policies .......................................... 86
  4.3.2 FIFO: Submission Order ............................................... 86
  4.3.3 Prioritizing Jobs by User, Project or Group ...................... 87
  4.3.4 Allocating Resources by User, Project or Group ............... 88
  4.3.5 Scheduling Jobs According to Size Etc ......................... 91
  4.3.6 Scheduling Jobs into Time Slots .................................... 93
Chapter 4  Scheduling

4.3.7 Default Scheduling Policy .................................................. 96
4.3.8 Examples of Workload and Policy ........................................ 98

4.4 The Scheduler ................................................................. 100
  4.4.1 Configuring the Scheduler ............................................. 101
  4.4.2 Making the Scheduler Read its Configuration ..................... 110
  4.4.3 Scheduling on Resources .............................................. 111
  4.4.4 Starting, Stopping, and Restarting the Scheduler ................. 111
  4.4.5 The Scheduling Cycle .................................................. 112
  4.4.6 How Available Consumable Resources are Counted ............... 113

4.5 Using Queues in Scheduling .............................................. 114

4.6 Scheduling Restrictions and Caveats .................................... 115
  4.6.1 One Policy Per PBS Complex ......................................... 115
  4.6.2 Jobs that Cannot Run on Current Resources ....................... 115
  4.6.3 Resources Not Controlled by PBS .................................. 115
  4.6.4 No Pinning of Processes to Cores ................................ 116

4.7 Errors and Logging .......................................................... 116
  4.7.1 Logfile for scheduler ................................................ 116

4.8 Scheduling Tools ............................................................. 116
  4.8.1 Anti-Express Queues .................................................. 120
  4.8.2 Associating Vnodes with Queues .................................... 121
  4.8.3 Using Backfilling ..................................................... 124
  4.8.4 Examining Jobs Queue by Queue .................................... 129
  4.8.5 Checkpointing ......................................................... 131
  4.8.6 Organizing Job Chunks ............................................... 131
  4.8.7 cron Jobs, or the Windows Task Scheduler ......................... 132
  4.8.8 Using Custom and Default Resources ............................... 133
  4.8.9 Using Idle Workstation Cycle Harvesting .......................... 136
  4.8.10 Dedicated Time ....................................................... 153
  4.8.11 Dependencies ......................................................... 154
  4.8.12 Dynamic Resources .................................................. 154
  4.8.13 Eligible Wait Time for Jobs ......................................... 155
  4.8.14 Sorting Jobs by Entity Shares (Was Strict Priority) ............ 160
  4.8.15 Estimating Job Start Time ........................................... 161
  4.8.16 Calculating Job Execution Priority ................................. 166
  4.8.17 Express Queues ....................................................... 169
  4.8.18 Using Fairshare ..................................................... 170
  4.8.19 FIFO Scheduling ..................................................... 180
  4.8.20 Using a Formula for Computing Job Execution Priority .......... 182
  4.8.21 Gating Jobs at Server or Queue .................................... 190
  4.8.22 Managing Application Licenses ..................................... 191
  4.8.23 Limits on Per-job Resource Usage ................................. 192
  4.8.24 Limits on Project, User, and Group Jobs .......................... 192
4.2 Scheduling Policy Basics

4.2.1 How Scheduling can be Used

You can use the scheduling tools provided by PBS to implement your chosen scheduling policy, to make your jobs run in the way you want.

Your policy can do the following:

• Prioritize jobs according to your specification
• Run jobs according to their relative importance
• Award specific amounts of resources such as CPU time or licenses to users and groups
4.2.2 What is Scheduling Policy?

Scheduling policy determines when each job is run and on which resources. In other words, a scheduling policy describes a goal, or intended behavior. For convenience, we describe a scheduling policy as being a combination of sub-goals, for example a combination of how resources should be allocated and how efficiency should be maximized.

You implement a scheduling policy using the tools PBS provides. A scheduling tool is a feature that allows you control over some aspect of scheduling. For example, the job sorting formula is a tool that allows you to define how you want job execution priority to be computed. Some scheduling tools are supplied by the PBS scheduler, and some are supplied by other elements of PBS, such as the server, queues or resources.

4.2.3 Basic PBS Scheduling Behavior

The basic behavior of PBS is that it always places jobs where it finds the resources requested by the job. PBS will not place a job where that job would use more resources than PBS thinks are available. For example, if you have two jobs, each requesting 1 CPU, and you have one vnode with 1 CPU, PBS will run only one job at a time on the vnode. You do not have to configure PBS for this basic behavior.

However, you do have to inform PBS which resources are available and where, how much, and whether they are consumable or not. In addition, in order to ensure that jobs are sent to the appropriate vnodes for execution, you also need to make sure that they request the correct resources. You can do this either by having users submit their jobs with the right resource requests, or by configuring default resources for jobs to inherit.

4.2.4 Sub-goals

Your scheduling policy is the combination that you choose of one or more sub-goals. For example, you might need to meet two particular sub-goals: you might need to prioritize jobs a certain way, and you might need to use resources efficiently. You can choose among various outcomes for each sub-goal. For example, you can choose to prioritize jobs according to size, owner, owner’s usage, time of submission, etc.
In the following sections, we describe the tools PBS offers for meeting each of the following sub-goals.

- Job prioritization; see section 4.2.5, “Job Prioritization”, on page 67.
- Resource allocation & limits; see section 4.2.6, “Resource Allocation to Users, Projects & Groups”, on page 74.
- Time slot allocation; see section 4.2.7, “Time Slot Allocation”, on page 77.
- Job placement optimizations; see section 4.2.8, “Job Placement Optimization”, on page 79.
- Resource efficiency optimizations; see section 4.2.9, “Resource Efficiency Optimizations”, on page 82.
- Overrides; see section 4.2.10, “Overrides”, on page 85.

### 4.2.5 Job Prioritization

Job prioritization is any technique you use to come up with a ranking of each job’s relative importance. You can specify separate priority schemes for both execution and preemption.

#### 4.2.5.1 Where PBS Uses Job Priority

PBS calculates job priority for two separate tasks: job execution and job preemption. Job execution priority is used with other factors to determine when to run each job. Job preemption priority is used to determine which queued jobs are allowed to preempt which running jobs in order to use their resources and run. These two tasks are independent, and it is important to make sure that you do not make them work at cross-purposes. For example, you do not want to have a class of jobs having high execution priority and low preemption priority; these jobs would run first, and then be preempted first.

Preemption comes into play when the scheduler examines the top job and determines that it cannot run now. If preemption is enabled, the scheduler checks to see whether the top job has sufficient preemption priority to be able to preempt any running jobs, and then if it does, whether preempting jobs would yield enough resources to run the top job. If both are true, the scheduler preempts running jobs and runs the top job.

If you take no action to configure how jobs should be prioritized, they are considered in submission order, one queue at a time. If you don’t prioritize queues, the queues are examined in an undefined order.
4.2.5.2 Overview of Prioritizing Jobs

PBS provides several tools (methods) for setting job execution priority. There are queue-based tools for organizing jobs, moving them around, and specifying the order in which groups of jobs should be examined. There are tools for sorting jobs into the order you want. There is a meta-tool (strict ordering) that allows you to specify that the top job must go next, regardless of whether the resources it requires are available now.

The scheduler can use only one sorting tool at a time. However, you can combine your chosen sorting tool with queue-based tools to give a wide variety of behaviors. Most of the queue-based tools can be used together. The scheduler can treat all jobs as if they are in a single queue, considering them all with respect to each other, or it can examine jobs queue by queue, comparing each job only to other jobs in the same queue.

You can change how execution priority is calculated, depending on which time slot is occurring. You can divide time up into primetime, non-primetime, and dedicated time.

When the scheduler calculates job execution priority, it uses a built-in system of job classes. PBS runs special classes of jobs before it considers queue membership. These classes are for reservation, starving, suspended, and express jobs. Please see section 4.8.16, “Calculating Job Execution Priority”, on page 166. After these jobs are run, the scheduler follows the rules you specify for queue behavior. Within each queue, jobs are sorted according to the sorting tool you choose.

4.2.5.3 Using Queue-based Tools to Prioritize Jobs

4.2.5.3.i Using Queue Order to Affect Order of Consideration

When the scheduler examines waiting jobs, it can either consider all of the jobs in the complex as a whole, or it can examine jobs in only one queue at a time. When examining jobs one queue at a time, the scheduler can either run all the jobs it can from the first queue before examining the jobs in the next queue (the default behavior), or run one job from each queue before moving to the next queue. These two methods of examining queues individually are incompatible, and both are incompatible with treating all jobs as if they are in a single queue.

The by_queue scheduler parameter controls whether or not the scheduler runs all the jobs it can from the first queue before moving to the next. By default, this parameter is set to True. When the round_robin parameter is True, it overrides by_queue. See section 4.8.4, “Examining Jobs Queue by Queue”, on page 129.

The round_robin scheduler parameter controls whether or not the scheduler examines one queue at a time, running one job from each queue before moving on to the next queue. This parameter overrides by_queue. See section 4.8.38, “Round Robin Queue Selection”, on page 255.
If you want queues to be considered in a specific order, you must assign a different priority to each queue, and tell PBS to sort the queues according to their priority. To sort the queues according to their priority, set the `sort_queues` scheduler parameter to `True`. The `sort_queues` parameter is deprecated as of 12.2. See section 4.8.45, “Sorting Queues into Priority Order”, on page 278.

Give the queue you want considered first the highest priority, then the next queue the next highest priority, and so on. If the queues don’t have priority assigned to them, the order in which they are considered is undefined. To set a queue’s priority, use the `qmgr` command to assign a value to the `priority` queue attribute. See section 2.2.5.3, “Prioritizing Execution Queues”, on page 23.

4.2.5.3.ii Using Express Queues in Job Priority Calculation

You can create express queues, and route jobs into them, if you want to give those jobs special priority.

An express queue is a queue whose priority is high enough to qualify as an express queue; the default for qualification is 150, but this can be set using the `preempt_queue_prio` scheduler parameter. For information on configuring express queues, see section 2.2.5.3.i, “Express Queues”, on page 24.

When calculating execution priority, the PBS scheduler uses a built-in job class called “Express” which contains all jobs that have a preemption level greater than that of the `normal_jobs` level. By default, those jobs are jobs in express queues. See section 4.8.16, “Calculating Job Execution Priority”, on page 166.

You can create preemption levels that include jobs in express queues. Jobs in higher preemption levels are allowed to preempt jobs in lower levels. See section 4.8.33, “Using Preemption”, on page 228.

4.2.5.3.iii Routing Jobs into Queues

You can configure PBS to automatically put each job in the most appropriate queue. There are several approaches to this. See section 4.8.39, “Routing Jobs”, on page 256.

4.2.5.3.iv Using Queue Priority when Computing Job Priority

You can configure the scheduler so that job priority is partly determined by the priority of the queue in which the job resides. See section 4.8.36, “Queue Priority”, on page 248.
### 4.2.5.4 Using Job Sorting Tools to Prioritize Jobs

The scheduler can use only one sorting tool at a time. The sorting tool you choose can be used either on all jobs in the complex together, or within each queue, one queue at a time. The job sorting tools are the following:

- You can create a formula that the scheduler uses to sort jobs. The scheduler applies this formula to all jobs in the complex, using it to calculate a priority for each job. For example, you can specify in the formula that jobs requesting more CPUs have higher priority. If the formula is defined, it overrides fairshare and sorting jobs on keys. See section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.

- You can use the fairshare algorithm to sort jobs. This algorithm allows you to set a resource usage goal for users or groups. Jobs are prioritized according to each entity’s usage; jobs whose owners have used the smallest percentage of their allotment go first. For example, you can track how much CPU time is being used, and allot each group a percentage of the total. See section 4.8.18, “Using Fairshare”, on page 170.

- You can sort jobs according to the same usage allotments you set up for fairshare. In this case, jobs whose owners are given the highest allotment go first. See section 4.8.14, “Sorting Jobs by Entity Shares (Was Strict Priority)”, on page 160.

- You can sort jobs on one or more keys, for example, you can sort jobs first by the number of CPUs they request, then by the amount of memory they request. You can specify that either the high or the low end of the resulting sort has higher priority.

  You can create a custom resource, and use a hook to set a value for that resource for each job, and then sort on the resource. See section 4.8.43, “Sorting Jobs on a Key”, on page 275.

- You can run jobs in the order in which they were submitted. See section 4.8.19, “FIFO Scheduling”, on page 180.

- You can run jobs according to the priority requested for each job at submission time. This priority can be modified via a hook. See section 4.8.44, “Sorting Jobs by Requested Priority”, on page 278 and Chapter 6, "Hooks", on page 413.

### 4.2.5.5 Prioritizing Jobs by Wait Time

You can use the amount of time a job has been waiting to run in the priority calculation. There are two ways to measure wait time:

- Eligible waiting time: how long a job has been waiting to run due to a shortage of resources, rather than because its owner isn’t allowed to run jobs now. See section 4.8.13, “Eligible Wait Time for Jobs”, on page 155

- Amount of time waiting in the queue
Both of these ways can be used when computing whether or not a job is starving. You can specify how long a job must be waiting to be considered starving. See section 4.8.46, “Starving Jobs”, on page 279.

You can use a job’s eligible waiting time in the job sorting formula. See section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.

When a job is considered to be starving, it is automatically assigned special execution priority, and placed in the Starving execution priority class; see section 4.8.16, “Calculating Job Execution Priority”, on page 166. You can configure preemption levels that include starving jobs; see section 4.8.33, “Using Preemption”, on page 228.

### 4.2.5.6 Calculating Preemption Priority

Execution priority and preemption priority are two separate systems of priority.

By default, if the top job cannot run now, and it has high preemption priority, the scheduler will use preemption to run the top job. The scheduler will preempt jobs with lower preemption priority so that it can use the resources to run the top job. The default for jobs with high preemption priority is jobs in express queues. You can configure many levels of preemption priority, specifying which levels can preempt which other levels. See section 4.8.33, “Using Preemption”, on page 228.

### 4.2.5.7 Meta-priority: Running Jobs Exactly in Priority Order

By default, when scheduling jobs, PBS orders jobs according to execution priority, then considers each job, highest-priority first, and runs the next job that can run now. If a job cannot run now because the resources required are unavailable, the default behavior is to skip the job and move to the next in order of priority.

You can tell PBS to use a different behavior called strict ordering. This means that you tell PBS that it must not skip a job when choosing which job to run. If the top job cannot run, no job runs.

You can see that using strict ordering could lead to decreased throughput and idle resources. In order to prevent idle resources, you can tell PBS to run small filler jobs while it waits for the resources for the top job to become available. These small filler jobs do not change the start time of the top job. See section 4.8.47, “Using Strict Ordering”, on page 282 and section 4.8.3, “Using Backfilling”, on page 124.
4.2.5.8 Using Different Calculations for Different Time Periods

PBS allows you to divide time into two kinds, called primetime and non-primetime. All time is covered by one or the other of these two kinds of time. The times are arbitrary; you can set them up however you like. You can also choose not to define them, and instead to treat all time the same.

You can configure two separate, independent ways of calculating job priority for primetime and non-primetime. The same calculations are used during dedicated time; dedicated time is a time slot made up of primetime and/or non-primetime. Many scheduler parameters are prime options, meaning that they can be configured separately for primetime and non-primetime. For example, you can configure fairshare as your sorting tool during primetime, but sort jobs on a key during non-primetime.

If you use the formula, it is in force all of the time.


4.2.5.9 When Priority Isn’t Enough: Overrides

Sometimes, the tools available for setting job priority don’t do everything you need. For example, it may be necessary to run a job right away, regardless of what else is running. Or you may need to put a job on hold. Or you might need to tweak the way the formula works for the next $N$ jobs. See section 4.8.30, “Overrides”, on page 201.

4.2.5.10 Elements to Consider when Prioritizing Jobs

- Whether users, groups, or projects affect job priority: for techniques to use user, group, or project to affect job priority, see section 4.3.3, “Prioritizing Jobs by User, Project or Group”, on page 87.
- Starving jobs: PBS has a built-in execution priority for starving jobs, but you can give starving jobs the highest execution priority by giving them the highest preemption priority and enabling preemption. See section 4.8.16, “Calculating Job Execution Priority”, on page 166 and section 4.8.33, “Using Preemption”, on page 228.
- Express jobs: PBS has a built-in execution priority for express jobs. You can set the preemption priority for express jobs; see section 4.8.33, “Using Preemption”, on page 228.
- Suspended jobs: PBS has a built-in execution priority for suspended jobs. You can set the preemption priority for suspended jobs; see section 4.8.33, “Using Preemption”, on page 228.
- Large or small jobs: you may want to give large and/or small jobs special treatment. See section 4.3.5, “Scheduling Jobs According to Size Etc.”, on page 91.
- User’s priority request for job: the job submitter can specify a priority for the job at sub-
mission. You can sort jobs according to each job’s specified priority. See section 4.8.44, “Sorting Jobs by Requested Priority”, on page 278.

- Whether the top job must be the next to run, regardless of whether it can run now; see section 4.8.47, “Using Strict Ordering”, on page 282.

4.2.5.10.i Queue-based Tools for Organizing Jobs

These tools can be used in conjunction with the sorting tools listed above.

- Queue-by-queue: PBS runs all the jobs it can from the first queue before moving to the next queue. Queue order is determined by priority and the value of the sort_queues parameter. The sort_queues parameter is deprecated as of 12.2. See section 4.8.4, “Examining Jobs Queue by Queue”, on page 129.


- Queue priority: Queues can be ordered according to their priority; jobs in higher-priority queues can then be examined before those in lower-priority queues. See section 2.2.5.3, “Prioritizing Execution Queues”, on page 23.

- Sorting queues: PBS can sort queues into priority order. See section 4.8.45, “Sorting Queues into Priority Order”, on page 278.

- Express queues: Jobs in express queues are assigned increased priority. See section 2.2.5.3.i, “Express Queues”, on page 24, and section 4.2.5.3.ii, “Using Express Queues in Job Priority Calculation”, on page 69.

- Routing: You can set up a queue system so that jobs with certain characteristics are routed to specific queues. See section 4.8.39, “Routing Jobs”, on page 256.

4.2.5.10.ii Job Sorting Tools

You can use only one job sorting tool at a time, and the tool you choose applies to all queues. However, you could use a different sorting tool for primetime and non-primetime. Each of these methods can be used on all jobs in the complex, or one queue at a time.

- Job sorting formula: You create a formula that PBS uses to calculate each job’s priority. See section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.


- Sorting jobs on keys: PBS can sort jobs according to one or more keys, such as requested CPUs or memory; see section 4.8.43, “Sorting Jobs on a Key”, on page 275.

- Entity shares (strict priority): Jobs are prioritized according to the owner’s fairshare allocation. See section 4.8.14, “Sorting Jobs by Entity Shares (Was Strict Priority)”, on page 182.
4.2.5.10.iii Other Job Prioritization Tools

- Strict ordering: you can specify that jobs must be run in priority order, so that a job that cannot run because resources are unavailable is not skipped. See section 4.8.47, “Using Strict Ordering”, on page 282.

- Waiting time: PBS can assign increased priority to jobs that have been waiting to run. See section 4.8.13, “Eligible Wait Time for Jobs”, on page 155, and section 4.8.46, “Starving Jobs”, on page 279.

- Setting job execution priority: PBS can set job execution priority according to a set of rules. See section 4.8.16, “Calculating Job Execution Priority”, on page 166.


- Starving jobs: Jobs that have been waiting for a specified amount of time can be given increased priority. See section 4.8.46, “Starving Jobs”, on page 279.

- Behavior overrides: you can intervene manually in how jobs are run. See section 4.8.30, “Overrides”, on page 201.

4.2.6 Resource Allocation to Users, Projects & Groups

If you need to ensure fairness, you may need to make sure that resources are allocated fairly. If different users, groups, or projects own or pay for different amounts of hardware or machine time, you may need to allocate resources according to these amounts or proportions.

You can allocate hardware-based resources such as CPUs or memory, and/or time-based resources such as walltime or CPU time, according to to the agreed amounts or proportions. You can also control who starts jobs.

4.2.6.1 Limiting Amount of Resources Used

4.2.6.1.i Allocation Using Resource Limits

You can use resource limits as a way to enforce agreed allocation amounts. This is probably the most straightforward way, and the easiest to explain to your users. PBS provides a system for limiting the total amount of each resource used by projects, users, and groups at the server and at each queue. For example, you can set a limit on the number of CPUs that any generic
user can use at one time at QueueA, but set three different individual limits for each of three users that have special requirements, at the same queue. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367.

4.2.6.1.ii Allocation Using Fairshare

The PBS fairshare tool allows you to start jobs according to how much of a particular resource has been used by job owners. You can designate who the valid job owners are, which resource is being tracked, and how much of the resource each owner is allowed to be using. Fairshare uses a moving average of resource usage, so that a user who in the recent past has not used their share can use more now. For example, you can track usage of the cpud resource, and give one group 40 percent of usage, one 50 percent, and one group, 10 percent. See section 4.8.18, “Using Fairshare”, on page 170.

4.2.6.1.iii Allocation Using Routing

If you do not want to place usage limits directly on projects, users, or groups, you can instead route their jobs to specific queues, where those queues have their own resource usage limits. To route jobs this way, force users to submit jobs to a routing queue, and set access control limits at each execution queue. See section 8.3, “Using Access Control”, on page 647. Make the routing queue be the default queue:

```bash
Qmgr: set server default_queue = <routing queue name>
```

Using this method, you place a limit for total resource usage at each queue, for each resource you care about. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367.

4.2.6.2 Limiting Jobs

4.2.6.2.i Limiting Number of Jobs per Project, User, or Group

You can set limits on the numbers of jobs that can be run by projects, users, and groups. You can set these limits for each project, user, and group, and you can set them at the server and at each queue. You can set a generic limit for all projects, users, or groups, and individual limits that override the generic limit. For example, you can set a limit that says that no user at the complex can run more than 8 jobs. Then you can set a more specific limit for QueueA, so that users at QueueA can run 4 jobs. Then you can set a limit for User1 and User2 at QueueA, so that they can run 6 jobs. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367.
4.2.6.2.ii  Allocation Using Round-robin Queue Selection

PBS can select jobs from queues by examining the queues in round-robin fashion. When using the round-robin method, the scheduler considers the first queue, tries to run a job from that queue, then considers the next queue, tries to run a job from that queue, then considers the next queue, and so on, in a circular fashion. If you want a simple way to control how jobs are started, you can use round-robin where each queue belongs to a different user or group. See section 4.8.38, “Round Robin Queue Selection”, on page 255.

4.2.6.2.iii  Limiting Resource Usage per Job

If you are having trouble with large jobs taking up too much of a resource, you can limit the amount of the resource being used by individual jobs. You can set these limits for each queue, and at the server. See section 5.15.3, “Placing Resource Limits on Jobs”, on page 391.

4.2.6.3  Resource Allocation Tools

The following is a list of scheduling tools that you can use for allocating resources or limiting resources or jobs:

• Matching: PBS places jobs where the available resources match the job’s resource requirements; see section 4.8.28, “Matching Jobs to Resources”, on page 197.

• Reservations: Users can create advance and standing reservations for specific resources for specific time periods. See section 4.8.37, “Advance and Standing Reservations”, on page 249.


• Routing: You can set up a queue system so that jobs with certain characteristics are routed to specific queues. See section 2.2.6, “Routing Queues”, on page 24 and section 4.8.39, “Routing Jobs”, on page 256.

• Limits on resource usage by projects, users, and groups: You can set limits on user and group resource usage. See section 4.8.25, “Limits on Project, User, and Group Resource Usage”, on page 192.

• Round-robin job selection: PBS can select jobs from queues in a round-robin fashion. See section 4.8.38, “Round Robin Queue Selection”, on page 255.

• Sorting queues: PBS can sort queues into priority order. See section 4.8.45, “Sorting Queues into Priority Order”, on page 278.

• Limits on number of jobs for projects, users, and groups: You can set limits on the numbers of jobs that can be run by users and groups. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367.

• Limits on resources used by each job: You can set limits on the amount of each resource
that any job can use. See section 4.8.23, “Limits on Per-job Resource Usage”, on page 192.

- Limit on number of jobs at each vnode: You can set limits on the number of jobs that can run at each vnode. See section 4.8.26, “Limits on Jobs at Vnodes”, on page 193.

- Using custom resources to limit resource usage: You use custom resources to manage usage. See section 4.8.8, “Using Custom and Default Resources”, on page 133.

- Gating and admission requirements: You can specify admission requirements for jobs. See section 4.8.21, “Gating Jobs at Server or Queue”, on page 190.

- Making jobs inherit default resources: You can use default resources to manage jobs. See section 4.8.8, “Using Custom and Default Resources”, on page 133.

4.2.7 Time Slot Allocation

Time slot allocation is the process of creating time slots within which only specified jobs are allowed to run.

4.2.7.1 Why Allocate Time Slots

You may want to set up blocks of time during which only certain jobs are allowed to run. For example, you might need to ensure that specific high-priority jobs have their own time slot, so that they are guaranteed to be able to run and finish before their results are required.

You may want to divide jobs into those that run at night, when no one is around, and those that run during the day, because their owners need the results then.

You might want to run jobs on desktop clusters only at night, when the primary users of the desktops are away.

When you upgrade PBS, a chunk of dedicated time can come in very handy. You set up dedicated time for a time period that is long enough for you to perform the upgrade, and you make sure the time slot starts far enough out that no jobs will be running.

You may want to run different scheduling policies at different times or on different days.

4.2.7.2 How to Allocate Time Slots

Time slots are controlled by queues: primetime queues, non-primetime queues, dedicated time queues, and reservation queues. Therefore, you can use your favorite routing method to move jobs into the desired queues. See section 4.8.39, “Routing Jobs”, on page 256.
4.2.7.2.i  Allocation Using Primetime and Holidays
You can specify how to divide up days or weeks, and designate each time period to be either primetime or non-primetime. You can use this division in the following ways:
• You can run a different policy during primetime from that of non-primetime
• You can run specific jobs during primetime, and others during non-primetime

4.2.7.2.ii  Allocation Using Dedicated Time
Dedicated time is a time period where the only jobs that are allowed to run are the ones in dedicated time queues. The policy you use during dedicated time is controlled by the normal primetime and non-primetime policies; those times overlap dedicated time.
If you don’t allow any jobs into a dedicated time queue, you can use it to perform an upgrade.

4.2.7.2.iii  Allocation Using Reservations
You and any other PBS user can create advance and standing reservations. These are time periods with a defined start and end, for a specific, defined set of resources. Reservations are used to make sure that specific jobs can run on time. See section 4.8.37, “Advance and Standing Reservations”, on page 249.

4.2.7.2.iv  Allocation Using cron Jobs or the Windows Task Scheduler
You can use cron or the Windows Task Scheduler to run jobs at specific times. See section 4.8.7, “cron Jobs, or the Windows Task Scheduler”, on page 132.

4.2.7.3  Time Slot Allocation Tools
The following is a list of scheduling tools that you can use to create time slots:
• Primetime and holidays: You can specify days and times that are to be treated as prime execution time. See section 4.8.34, “Using Primetime and Holidays”, on page 241.
• Dedicated time: You can set aside blocks of time reserved for certain operations. See section 4.8.30.6, “Using Dedicated Time”, on page 205.
• cron jobs and the Windows Task Scheduler: You can use cron or the Windows Task Scheduler to run jobs. See section 4.8.30.7, “Using cron Jobs or the Windows Task Scheduler”, on page 205.
• Reservations: Users can create advance and standing reservations for specific resources for specific time periods. See section 4.8.37, “Advance and Standing Reservations”, on page 249.
### 4.2.8 Job Placement Optimization

PBS automatically places jobs where they can run, but you can refine how jobs are placed. Optimizations are the techniques you use to increase throughput, turnaround, or efficiency, by taking advantage of where jobs can be run.

PBS places jobs according to placement optimization settings in tools to specify how vnodes should be organized, how jobs should be distributed, and how resources should be used.

#### 4.2.8.1 Why Optimize Placement

PBS automatically places jobs where they can run, matching jobs to resources, so why optimize placement?

- You can help PBS refine its understanding of hardware topology, so that PBS can place jobs where they will run most efficiently.
- If you have some vnodes that are faster than others, you can preferentially place jobs on those vnodes.
- You may need to place jobs according to machine ownership, so that for example only jobs owned by a specific group run on a particular machine.
- You can take advantage of unused workstation computing capacity.
- You can balance the workload between two or more PBS complexes, trading jobs around depending on the workload on each complex.
- You can specify whether or not certain vnodes should be used for more than one job at a time.
- You can tell PBS to avoid placing jobs on highly-loaded vnodes

#### 4.2.8.2 Matching Jobs to Resources

By default, PBS places jobs where the available resources match the job’s resource requirements. See section 4.8.28, “Matching Jobs to Resources”, on page 197.

#### 4.2.8.3 Organizing and Selecting Vnodes

By default, the order in which PBS examines vnodes is undefined. The default setting for vnode sorting is the following:

```
node_sort_key: “sort_priority HIGH all”
```

However, sort_priority means sort on each vnode’s priority attribute, but by default, that attribute is unset.
By default, PBS does not organize vnodes into groups.

By default, when PBS chooses vnodes for a job, it runs down its list of vnodes, searching until it finds vnodes that can supply the job with the requested resources. You can improve this in two ways:

- PBS provides a way to organize your vnodes so that jobs can run on groups of vnodes, where the selected group of vnodes provides the job with good connectivity. This can improve memory access and interprocess communication speed. PBS then searches through these groups of vnodes, called placement sets, looking for the smallest group that satisfies the job’s requirements. Each placement set is a group of vnodes that share a value for a resource. An illustrative example is a group of vnodes that are all connected to the same high speed switch, so that all of the vnodes have the same value for the switch resource. For detailed information on how placement sets work and how to configure them, see section 4.8.32, “Placement Sets”, on page 211.

- By default, the order in which PBS examines vnodes, whether in or outside of placement sets, is undefined. PBS can sort vnodes on one or more keys. Using this tool, you can specify which vnodes should be selected first. For information on sorting vnodes on keys, see section 4.8.48, “Sorting Vnodes on a Key”, on page 283.

You can sort vnodes in conjunction with placement sets.

### 4.2.8.4 Distributing Jobs

All of the following methods for distributing jobs can be used together.

#### 4.2.8.4.i Filtering Jobs to Specific Vnodes

If you want to run certain kinds of jobs on specific vnodes, you can route those jobs to specific execution queues, and tie those queues to the vnodes you want. For example, if you want to route jobs requesting large amounts of memory to your large-memory machines, you can set up an execution queue called LMemQ, and associate that queue with the large-memory vnodes. You can route any kind of job to its own special execution queue. For example, you can route jobs owned by the group that owns a cluster to a special queue which is associated with the cluster. For details on routing jobs, see section 4.8.39, “Routing Jobs”, on page 256. For details on associating vnodes and queues, see section 4.8.2, “Associating Vnodes with Queues”, on page 121.

#### 4.2.8.4.ii Running Jobs at Least-loaded Complex

You can set up cooperating PBS complexes that automatically run jobs from each other’s queues. This allows you to dynamically balance the workload across multiple, separate PBS complexes. See section 4.8.31, “Peer Scheduling”, on page 205.
4.2.8.4.iii Using Idle Workstations
You can run jobs on workstations whenever they are not being used by their owners. PBS can monitor workstations for user activity or load, and run jobs when those jobs won’t interfere with the user’s operation. See section 4.8.9, “Using Idle Workstation Cycle Harvesting”, on page 136.

4.2.8.4.iv Avoiding Highly-loaded Vnodes
You can tell PBS not to run jobs on vnodes that are above a specified load. This is in addition to the default behavior, where PBS does not run jobs that request more of a resource than it thinks each vnode can supply. See section 4.8.27, “Using Load Balancing”, on page 193.

4.2.8.4.v Placing Job Chunks on Desired Hosts
You can tell PBS to place each job on as few hosts as possible, to place each chunk of a job on a separate host, a separate vnode, or on any vnode. You can specify this behavior for the jobs at a queue and at the server.
You can do the following
• Set default behavior for the queue or server: jobs inherit behavior if they do not request it; see section 5.9.3.6, “Specifying Default Job Placement”, on page 307
• Use a hook to set each job’s placement request (Resource_List.place). See Chapter 6, "Hooks”, on page 413
For more on placing chunks, see section 4.8.6, “Organizing Job Chunks”, on page 131.
For information on how jobs request placement, see section 2.59.2.5, “Requesting Resources and Placing Jobs”, on page 214.

4.2.8.5 Shared or Exclusive Resources and Vnodes
PBS can give jobs their own vnodes, or fill vnodes with as many jobs as possible. The scheduler uses a set of rules to determine whether a job can share resources or a host with another job. These rules specify how the vnode sharing attribute should be combined with a job’s placement directive. The vnode’s sharing attribute supersedes the job’s placement request.
You can set each vnode’s sharing attribute so that the vnode or host is always shared, always exclusive, or so that it honors the job’s placement request. See section 4.8.40, “Shared vs. Exclusive Use of Resources by Jobs”, on page 261.

4.2.8.6 Tools for Organizing Vnodes
• Placement sets: PBS creates sets of vnodes organized by the values of multiple resources.
4.2.8.7 Tools for Distributing Jobs

- Routing: You can set up a queue system so that jobs with certain characteristics are routed to specific queues. See section 2.2.6, “Routing Queues”, on page 24 and section 4.8.39, “Routing Jobs”, on page 256.
- Associating vnodes with queues: You can specify that jobs in a given queue can run only on specific vnodes, and vice versa. See section 4.8.2, “Associating Vnodes with Queues”, on page 121.
- Load balancing: PBS can place jobs so that machines have balanced loads. See section 4.8.27, “Using Load Balancing”, on page 193.

4.2.9 Resource Efficiency Optimizations

PBS automatically runs each job where the resources required for the job are available. You can refine the choices PBS makes.

Resource optimizations are the techniques you use to increase throughput, turnaround, or efficiency, by taking advantage of how resources are used.

Before reading this section, please make sure you understand how resources are used by reading section 4.8.28, “Matching Jobs to Resources”, on page 197.

4.2.9.1 Why Optimize Use of Resources

You may want to take advantage of the following:
- If you are using strict ordering, you can prevent resources from standing idle while the
top job waits for its resources to become available

- PBS can estimate the start times of jobs, so that users can stay informed
- PBS can provision vnodes with the environments that jobs require
- PBS can track resources that are outside of the control of PBS, such as scratch space
- You can take advantage of unused workstation computing capacity
- You can balance the workload between two or more PBS complexes, trading jobs around depending on the workload on each complex.
- You can specify whether or not certain vnodes should be used for more than one job at a time.
- Users can specify that jobs that are dependent on the output of other jobs run only after the other jobs complete
- You can tell PBS to avoid placing jobs on highly-loaded vnodes

### 4.2.9.2 How to Optimize Resource Use

#### 4.2.9.2.i Backfilling Around Top Jobs

PBS creates a list of jobs ordered by priority, and tries to run the jobs in order of priority. You can force all jobs to be run in exact order of their priority, using strict ordering. See section 4.8.47, “Using Strict Ordering”, on page 282. However, this can reduce resource utilization when the top job cannot run now and must wait for resources to become available. You can offset this problem by using backfilling, where PBS tries to fit smaller jobs in around the top job that cannot run. The start time of the top job is not delayed. Job walltimes are required in order to use backfilling. You can specify the number of jobs around which to backfill. You can also disable this feature. See section 4.8.3, “Using Backfilling”, on page 124.

PBS can shrink the walltime of shrink-to-fit jobs into available time slots. These jobs can be used to backfill around top jobs and time boundaries such as dedicated time or reservations. See section 4.8.41, “Using Shrink-to-fit Jobs”, on page 263.

If you do not use strict ordering, PBS won’t necessarily run jobs in exact priority order. PBS will instead run jobs so that utilization is maximized, while trying to preserve priority order.

#### 4.2.9.2.ii Using Dependencies

Job submitters can specify the dependencies between jobs. For example, if you have an end-of-day accounting job that must run after certain other jobs, you can specify that. See section 4.8.11, “Dependencies”, on page 154.
4.2.9.2.iii Estimating Start Time for Jobs

You can tell PBS to estimate start times and execution vnodes for either the number of jobs being backfilled around, or all jobs. Users can then see when their jobs are estimated to start, and the vnodes on which they are predicted to run. See section 4.8.15, “Estimating Job Start Time”, on page 161.

4.2.9.2.iv Provisioning Vnodes with Required Environments

PBS can provision vnodes with environments (applications or operating systems) that jobs require. This means that a job can request a particular environment that is not yet on a vnode, but is available to be instantiated there. See section 4.8.35, “Provisioning”, on page 247.

4.2.9.2.v Tracking Dynamic Resources

You can use dynamic PBS resources to represent elements that are outside of the control of PBS, typically for licenses and scratch space. You can represent elements that are available to the entire PBS complex as server-level resources, or elements that are available at a specific host or hosts as host-level resources. For an example of configuring a server-level dynamic resource, see section 5.14.4.1.i, “Example of Configuring Dynamic Server-level Resource”, on page 335. For an example of configuring a dynamic host-level resource, see section 5.14.5.1.i, “Example of Configuring Dynamic Host-level Resource”, on page 338.

For a complete description of how to create and use dynamic resources, see section 5.14, “Custom Resources”, on page 319.

4.2.9.3 Optimizing Resource Use by Job Placement

4.2.9.3.i Sending Jobs to Complex Having Lightest Workload

You can set up cooperating PBS complexes that automatically run jobs from each other’s queues. This allows you to dynamically balance the workload across multiple, separate PBS complexes. See section 4.8.31, “Peer Scheduling”, on page 205.

4.2.9.3.ii Using Idle Workstations

You can run jobs on workstations whenever they are not being used by their owners. PBS can monitor workstations for user activity or load, and run jobs when those jobs won’t interfere with the user’s operation. See section 4.8.9, “Using Idle Workstation Cycle Harvesting”, on page 136.

4.2.9.3.iii Avoiding Highly-loaded Vnodes

You can tell PBS not to run jobs on vnodes that are above a specified load. This is in addition to the default behavior, where PBS does not run jobs that request more of a resource than it thinks each vnode can supply. See section 4.8.27, “Using Load Balancing”, on page 193.
4.2.9.4 Resource Efficiency Optimization Tools

The following is a list of scheduling tools that you can use to optimize how resources are used:

- Backfilling around most important job(s): PBS can place small jobs in otherwise-unused blocks of resources. See section 4.8.3, “Using Backfilling”, on page 124.

- Dependencies: Users can specify requirements that must be met by previous jobs in order for a given job to run. See section 4.8.11, “Dependencies”, on page 154.

- Estimating start time of jobs: PBS can estimate when jobs will start, so that users can be informed. See section 4.8.15, “Estimating Job Start Time”, on page 161.

- Provisioning vnodes with required environments: PBS can provision vnodes with the environments that jobs require. See section 4.8.35, “Provisioning”, on page 247.

- Using dynamic resources: PBS can track resources such as scratch space and licenses. See section 4.8.12, “Dynamic Resources”, on page 154.


- Load balancing: PBS can place jobs so that machines have balanced loads. See section 4.8.27, “Using Load Balancing”, on page 193.

4.2.10 Overrides

Overrides are the techniques you use to override the specified scheduling behavior of PBS.

4.2.10.1 Why and How to Override Scheduling

- If you need to run a job immediately, you can tell PBS to run a job now. You can optionally specify the vnodes and resources to run it. See section 4.8.30.1, “Run a Job Manually”, on page 201.

- If you need to prevent a job from running, you can tell PBS to place a hold on a job. See section 4.8.30.2, “Hold a Job Manually”, on page 203.

- If you need to change how the formula computes job priority, you can make on-the-fly changes to how the formula is computed. See section 4.8.30.5, “Change Formula On the Fly”, on page 204.

- If you need a block of time where you can control what’s running, for example for upgrading PBS, you can create dedicated time. See section 4.8.30.6, “Using Dedicated
• If you need to submit jobs at a certain time, you can use `cron` or the Windows Task Scheduler to run jobs. See section 4.8.30.7, “Using `cron` Jobs or the Windows Task Scheduler”, on page 205.

• If you need to change job resource requests or attributes, you can use hooks to examine jobs and alter their characteristics. See Chapter 6, "Hooks", on page 413.

4.3 Choosing a Policy

4.3.1 Overview of Kinds of Policies

You can tune PBS to produce a wide variation in scheduling behavior. You can choose from a wide variety of behaviors for each sub-goal, resulting in many possible scheduling policies. However, policies can be grouped into the following kinds:

• FIFO, where you essentially run jobs in the order in which they were submitted; see section 4.3.2, “FIFO: Submission Order”, on page 86

• According to user or group priority, where the job’s priority is determined by the owner’s priority; see section 4.3.3, “Prioritizing Jobs by User, Project or Group”, on page 87

• According to resource allocation rules, where jobs are run so that they use resources following a set of rules for how resources should be awarded to users or groups; see section 4.3.4, “ Allocating Resources by User, Project or Group”, on page 88

• According to the size of the job, for example measured by CPU or memory request; see section 4.3.5, “Scheduling Jobs According to Size Etc.”, on page 91

• By setting up time slots for specific uses; see section 4.3.6, “Scheduling Jobs into Time Slots”, on page 93

4.3.2 FIFO: Submission Order

If you want jobs to run in the order in which they are submitted, use FIFO. You can use FIFO across the entire complex, or within each queue.

If it’s important that jobs run exactly in submission order, use FIFO with strict ordering. However, if you don’t want resources to be idle while the top job is stuck, you can use FIFO with strict ordering and backfilling.

To run jobs in submission order, selecting queues in order of their priority, see section 4.8.19.1, “Configuring Basic FIFO Scheduling”, on page 181 and section 4.8.19.3, “Queue by Queue FIFO”, on page 181.

To run jobs in submission order, with strict ordering, see section 4.8.19.4, “FIFO with Strict Ordering”, on page 181.

To run jobs in submission order, with strict ordering and backfilling, see section 4.8.19.5, “FIFO with Strict Ordering and Backfilling”, on page 182.

### 4.3.3 Prioritizing Jobs by User, Project or Group

If you need to run jobs from some users, groups, or projects before others, you can prioritize the jobs this way.

You can prioritize jobs by user, project, or group, using the following techniques:

- **Routing each entity’s jobs to its own execution queue, assigning the queue the desired priority, sorting the queues by priority, and examining jobs queue by queue.** See the following:
  - For routing: section 2.2.6, “Routing Queues”, on page 24
  - For setting queue priority: section 2.2.5.3, “Prioritizing Execution Queues”, on page 23
  - For sorting queues: section 4.8.45, “Sorting Queues into Priority Order”, on page 278
  - For examining jobs queue by queue: section 4.8.4, “Examining Jobs Queue by Queue”, on page 129

- **Routing each entity’s jobs to its own execution queue, where the jobs inherit a custom resource that you use in the job sorting formula.** See the following:
  - For routing: section 2.2.6, “Routing Queues”, on page 24
  - For inherited resources: section 11.2, “Allocating Resources to Jobs”, on page 821
  - For the job sorting formula: section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182

- **Using a hook to allocate a custom resource to each job, where the hook sets the value according to the priority of the job’s owner, group, or project, then using the resource in the job sorting formula.** See the following:
  - For hooks: Chapter 6, "Hooks", on page 413
  - For custom resources: section 5.14, “Custom Resources”, on page 319
  - For the job sorting formula: section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182
• Assigning a greater fairshare allocation in the fairshare tree to the users or groups whose jobs must run first, and running jobs according to entity shares. See the following:
  - For fairshare: section 4.8.18, “Using Fairshare”, on page 170
  - For entity shares: section 4.8.14, “Sorting Jobs by Entity Shares (Was Strict Priority)”, on page 160

4.3.4 Allocating Resources by User, Project or Group

When you want to divide up hardware usage among users, groups, or projects, you can make sure you allocate resources along those lines. You can do this in the following ways:

• Allocate portions of the entire complex to each entity; see section 4.3.4.1, “Allocating Portions of Complex”, on page 88

• Allocate portions of all machines or clusters to each entity, or use controlled allocation for some hardware, with a free-for-all elsewhere; see section 4.3.4.2, “Allocating Portions of Machines or Clusters”, on page 89

• Lock entities into using specific hardware; see section 4.3.4.3, “Locking Entities into Specific Hardware”, on page 90

4.3.4.1 Allocating Portions of Complex

4.3.4.1.i Allocating Specific Amounts

To allocate specific amounts of resources across the entire complex, you can use resource limits at the server. These limits set the maximum amount that can be used, ensuring that projects, users, or groups stay within their bounds. You can set a limit for each resource, and make it different for each project, user, and group. You can set a different limit for each project, user, and group, for each resource.

For example, you can set a limit of 48 CPUs in use at once by most groups, but give groupA a limit of 96 CPUs. You can give each individual user a limit of 8 CPUs, but give UserA a limit of 10 CPUs, and UserB a limit of 4 CPUs.

To set limits for usage across the entire complex, set the limits at the server.

4.3.4.1.ii Allocating Percentages

To allocate a percentage of the resources being used at the complex, you can use fairshare. Fairshare tracks a moving average of resource usage, so it takes past use into account. You choose which resource to track. You can tune the influence of past usage. To use fairshare, you write a file that contains the percentage of resource usage for each entity.

To use fairshare across the entire complex, make sure that both by_queue and round_robin are False.

Fairshare is described in section 4.8.18, “Using Fairshare”, on page 170.

4.3.4.2 Allocating Portions of Machines or Clusters

You can allocate fixed amounts of a machine or groups of machines. You can do this for as many machines as you want. For example, on HostA, you can give GroupA 100 CPUs, GroupB 150 CPUs, and GroupC 50 CPUs, while at HostB, GroupA gets 10, GroupB gets 8, and GroupC gets 25.

To allocate fixed portions of a specific machine or group of machines, you use these tools in combination:

• Create an execution queue for this machine; see section 2.2.3, “Creating Queues”, on page 21.

• Route jobs belonging to the users or groups who share this machine into a queue. Each machine or cluster that requires controls gets its own queue. See section 4.8.39, “Routing Jobs”, on page 256.

• Associate the queue with the vnodes in question; see section 4.8.2, “Associating Vnodes with Queues”, on page 121.

• Set a limit at the queue for each resource that you care about, for each project, user, or group. These limits control use of the vnodes associated with the queue only. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367.

You can prevent unauthorized usage by setting generic project, user, and group limits for the machine’s queue to zero. However, you probably don’t want users to submit their jobs to a queue where they are not allowed to run, only to have those jobs languish. You can avoid this by doing the following:

• Setting up a routing queue; see section 2.2.6, “Routing Queues”, on page 24.

• Make the routing queue be the default queue:

  Qmgr: set server default_queue = <routing queue name>

• Making the routing queue the only queue that accepts job submission: set from_route_only to True on execution queues tied to hardware. See section 2.2.5.1.
Chapter 4  Scheduling

“Where Execution Queues Get Their Jobs”, on page 22.

• Using queue access control to limit which jobs are routed into the execution queue; see section 2.2.6.5, “Using Access Control to Route Jobs”, on page 30.

You can either set up allocations for every machine, or you can set up allocations for only some machines, leaving a free-for-all for the others. If you want access to be unrestricted for some machines, do not set limits at the server.

4.3.4.3  Locking Entities into Specific Hardware

You can send all jobs from some projects, users, or groups to designated hardware; you are essentially limiting them to a sandbox. To do this, do the following:

• Create an execution queue for the sandbox hardware; see section 2.2.3, “Creating Queues”, on page 21.

• Create at least one other execution queue; see section 2.2.3, “Creating Queues”, on page 21.

• Create a routing queue; see section 2.2.3, “Creating Queues”, on page 21.

• Make the routing queue be the default queue:

Qmgr: set server default_queue = <routing queue name>

• Force all users to submit jobs to the routing queue: set from_route_only to True on all other queues. See section 2.2.5.1, “Where Execution Queues Get Their Jobs”, on page 22.

• Use queue access control to route according to user or group: allow jobs from the controlled users or groups into the sandbox queue only. See section 2.2.6.5, “Using Access Control to Route Jobs”, on page 30.

• Use a job submission hook to route according to project: route only the jobs from the desired project(s) to the sandbox queue. See Chapter 6, "Hooks", on page 413.

• Associate the sandbox queue with the sandbox vnodes. See section 4.8.2, “Associating Vnodes with Queues”, on page 121.

Note that you can either allow all projects, users, or groups into the sandbox queue, or allow only the controlled projects, users, or groups into the sandbox queue.
4.3.5 Scheduling Jobs According to Size Etc.

You may need to treat jobs differently depending on their size or other characteristics. For example, you might want to run jobs differently depending on the number of CPUs or amount of memory requested by the job, or whether the job requests GPUs.

- Give special priority to a group of jobs
- Run a group of jobs on designated hardware
- Run a group of jobs in designated time slots: reservations, dedicated time, and primetime or non-primetime

There are two main approaches to doing this. You can route jobs into queues, or you can use hooks to set values. Here is an outline:

- Route certain kinds of jobs into their own queues, in order to treat each kind differently. This works for priority, hardware, and time slots. See section 4.3.5.1, “Special Treatment via Routing”, on page 91.
  - Route each kind to its own queue, using queue-based routing or a submission hook;
  - Use queue-based methods to set job priority or to run the jobs on certain hardware or in certain time slots
- Use hooks to set priority for jobs or to set a custom resource that will send jobs to certain hardware. This does not work for time slots. See section 4.3.5.2, “Special Treatment via Hooks”, on page 93.
  - Use a submission hook to set each job’s Priority attribute, or set a value for a custom resource used in the job sorting formula
  - Use a submission hook to set a custom host-level resource value for each job, where the value matches the value at the desired hardware

4.3.5.1 Special Treatment via Routing

Use a routing queue or a hook to route jobs into a special queue, where the jobs are given special priority, or are run on special hardware, or are run in special time slots.

4.3.5.1.i Routing via Queues

- Create your destination queues. See section 2.2.3, “Creating Queues”, on page 21.
- Set limits at the destination queues, so that each queue receives the correct jobs. See section 2.2.6.4, “Using Resources to Route Jobs Between Queues”, on page 25.
- Create a routing queue, and set its destination queues. See section 2.2.6, “Routing”.

PBS Professional 12.2 Administrator’s Guide 91
Chapter 4  

Scheduling

- Make the routing queue be the default queue:

  `qmgr: set server default_queue = <routing queue name>`

4.3.5.1.ii  Using Hooks to Route Jobs

You can use a submission hook to move jobs into the queues you want. See section 4.8.39.2.ii, “Hooks as Mechanism to Move Jobs”, on page 259.

4.3.5.1.iii  Giving Routed Jobs Special Priority

You can give routed jobs special priority in the following ways:

- Have the jobs inherit a custom resource from the special queue, and use this resource in the job sorting formula.
  - For how to have jobs inherit custom resources, see section 11.2, “Allocating Resources to Jobs”, on page 821.
  - For how to use the job sorting formula, see section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.

- Give the queue itself special priority, and use queue priority in the job sorting formula.
  - For how to assign priority to queues, see section 2.2.5.3, “Prioritizing Execution Queues”, on page 23.
  - For how to use the job sorting formula, see section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.

4.3.5.1.iv  Running Jobs on Special Vnodes

Now that the special jobs are routed to a special queue, associate that queue with the special vnodes. See section 4.8.2, “Associating Vnodes with Queues”, on page 121.

4.3.5.1.v  Running Jobs in Special Time Slots

If you want to run jobs during dedicated time, route the jobs into one or more dedicated time queues. In the same way, for primetime or non-primetime, route jobs into primetime or non-primetime queues. You can also route jobs into reservation queues for reservations that you have created for this purpose.

For using dedicated time, see section 4.8.10, “Dedicated Time”, on page 153

For using primetime and non-primetime, see section 4.8.34, “Using Primetime and Holidays”, on page 241

For using reservations, see section 4.8.37, “Advance and Standing Reservations”, on page 249
4.3.5.2 Special Treatment via Hooks

4.3.5.2.i Setting Job Priority Via Hook

Use a job submission hook to set the job priority, by doing one of the following:

- Set a custom numeric resource for the job, and use the resource in the job sorting formula
  - For how to use hooks, see Chapter 6, "Hooks", on page 413
  - For how to use the job sorting formula, see section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.

- Set the job’s Priority attribute, and sort jobs on a key, where the key is the job’s Priority attribute.
  - For how to set job attributes, see Chapter 6, "Hooks", on page 413
  - For how to sort jobs on a key, see section 4.8.43, “Sorting Jobs on a Key”, on page 275.

4.3.5.2.ii Routing Jobs to Hardware via Hook

You can send jobs to particular hardware without using a particular queue, by using a hook. See section 4.8.39.4.i, “Using Hooks to Tag Jobs”, on page 261.

4.3.6 Scheduling Jobs into Time Slots

You can schedule jobs in time slots in the following ways:

- Set aside time slots for specific entities; see section 4.3.6.1, “Setting Aside Time Slots for Entities”, on page 93
- Lock entities into specific time slots; see section 4.3.6.2, “Locking Entities into Time Slots”, on page 94

4.3.6.1 Setting Aside Time Slots for Entities

You can set aside time slots that are reserved exclusively for certain users or groups. You can use reservations, dedicated time, or non-primetime.

4.3.6.1.i Reservations

Reservations set aside one or more blocks of time on the requested resources. Users can create their own reservations, or you can create them and set their access control to allow only specified users to submit jobs to them. See section 4.8.37, “Advance and Standing Reservations”, on page 249.
Chapter 4  Scheduling

4.3.6.1.ii  Dedicated Time

During dedicated time, the only jobs allowed to run are those in dedicated queues. The drawback to dedicated time is that it applies to the entire complex. If you want to set aside one or more time slots for a user or group, do the following:

- Create a dedicated queue. See section 2.2.5.2.i, “Dedicated Time Queues”, on page 23.
- Set access control on the dedicated queue so that only the particular users or groups you want can submit jobs to the queue. See section 2.2.6.5, “Using Access Control to Route Jobs”, on page 30.
- If you want to limit access on a dedicated queue to a specific project, set the generic limit for queued jobs for projects at that queue to zero, and then set the individual limit for the specific project higher.

4.3.6.1.iii  Non-primetime

You can set up primetime and non-primetime so that one of them, for example, non-primetime, is used as a special time slot allocated to particular users or groups. The advantage of using non-primetime is that you can set up a separate scheduling policy for it, for example, using fairshare during non-primetime and sorting jobs on a key during primetime. Note that the formula, if defined, is in force all of the time. To use non-primetime, do the following:

- Create a non-primetime queue; see section 2.2.3, “Creating Queues”, on page 21 and section 2.2.5.2.ii, “Primetime and Non-Primetime Queues”, on page 23.
- Define primetime and non-primetime; see section 4.8.34, “Using Primetime and Holidays”, on page 241.
- Set access control on the non-primetime queue so that only the particular users or groups you want can submit jobs to the queue. See section 2.2.6.5, “Using Access Control to Route Jobs”, on page 30.
- Make sure that the scheduling policy you want is in force during non-primetime. See section 4.8.34.1, “How Primetime and Holidays Work”, on page 241.

4.3.6.2  Locking Entities into Time Slots

You can make all jobs from some users or groups run during designated time slots. You can run them during a reservation, dedicated time, or non-primetime.

4.3.6.2.i  Locking Entities into Reservations

To allow a user to submit jobs only into a reservation, do the following:

- Create a reservation for the resources and time(s) you want the controlled user(s) to use.
When creating the reservation, set access control to allow the controlled user(s). See section 4.8.37, “Advance and Standing Reservations”, on page 249 and section 8.3.8.1, “Setting Reservation Access”, on page 660.

• Set access control on all queues except the reservation’s queue to deny the controlled user(s); see section 2.2.6.5, “Using Access Control to Route Jobs”, on page 30.

4.3.6.2.ii Locking Entities into Dedicated Time

You can create a dedicated time queue, and send all jobs from controlled projects, users, or groups to that queue. You can route their jobs to it, and you can allow them to submit directly to it. To lock one or more projects, users, or groups into one or more dedicated time slots, do the following:

• Create a dedicated time queue; see section 2.2.3, “Creating Queues”, on page 21 and section 2.2.5.2.1, “Dedicated Time Queues”, on page 23.

• Create at least one other execution queue; see section 2.2.3, “Creating Queues”, on page 21.

• Create a routing queue; see section 2.2.3, “Creating Queues”, on page 21.

• Prevent controlled users from submitting to non-dedicated time execution queues: set from_route_only to True on the non-dedicated time execution queues. See section 2.2.5.1, “Where Execution Queues Get Their Jobs”, on page 22.

• Use queue access control to allow jobs from the controlled users or groups into the dedicated time queue only. See section 2.2.6.5, “Using Access Control to Route Jobs”, on page 30.

• Use a job submission hook to route jobs from controlled projects into the dedicated time queue. See Chapter 6, “Hooks”, on page 413.

• Make the routing queue be the default queue:

  Qmgr: set server default_queue = <routing queue name>

Note that you can either allow all users into the dedicated time queue, or allow only the controlled users into the dedicated time queue.

4.3.6.2.iii Locking Entities into Non-primetime

You can create a non-primetime queue, and send all jobs from controlled users, groups, or projects to that queue. You can route their jobs to it, and you can allow them to submit directly to it. To lock one or more users, groups, or projects into one or more non-primetime slots, do the following:

• Create a non-primetime queue; see section 2.2.3, “Creating Queues”, on page 21 and section 2.2.5.2.2, “Primetime and Non-Primetime Queues”, on page 23.

• Create at least one other execution queue; see section 2.2.3, “Creating Queues”, on page 21.
Chapter 4  Scheduling

21.

• Create a routing queue; see section 2.2.3, “Creating Queues”, on page 21.

• Prevent controlled users from submitting to primetime execution queues: set from_route_only to True on the primetime execution queues. See section 2.2.5.1, “Where Execution Queues Get Their Jobs”, on page 22.

• Make the routing queue be the default queue:

```bash
qmgr: set server default_queue = <routing queue name>
```

• Use queue access control to allow jobs from the controlled users or groups into the non-primetime queue only. See section 2.2.6.5, “Using Access Control to Route Jobs”, on page 30.

• Use a job submission hook to route jobs from controlled projects into the non-primetime queue. See Chapter 6, “Hooks”, on page 413.

• Define primetime and non-primetime; see section 4.8.34, “Using Primetime and Holidays”, on page 241.

• Make sure that the scheduling policy you want is in force during non-primetime. See section 4.8.34.1, “How Primetime and Holidays Work”, on page 241.

Note that you can either allow all users into the non-primetime queue, or allow only the controlled users into the non-primetime queue.

4.3.7 Default Scheduling Policy

The default scheduling policy is determined by the default settings for all of the attributes, parameters, etc. that determine the scheduler’s behavior. For a list of all of these elements, see section 4.4.1, “Configuring the Scheduler”, on page 101.

The default behavior of the scheduler is the following:

• The scheduler matches jobs with available resources. This means that the scheduler places each job only where that job has enough resources to run. See section 4.8.28, “Matching Jobs to Resources”, on page 197.

• The scheduler will not over-allocate the resources that are listed in the scheduler’s resources parameter. The defaults for these are ncpus, mem, arch, host, vnode, aoe, netwins. See section 4.8.28.1, “Scheduling on Consumable Resources”, on page 197.

• The scheduler sorts vnodes according to its node_sort_key parameter, whose default set-
Scheduling

Node sorting is the following:

```
node_sort_key: "sort_priority HIGH all"
```

This means that vnodes are sorted by the value of their `priority` attribute, with high-priority vnodes used first. The scheduler places jobs first on vnodes that are first in the sorted list.

Note that all vnodes have the same default priority upon creation, so the default sorted order for vnodes is undefined.

See section 4.8.48, “Sorting Vnodes on a Key”, on page 283.

- Queues are sorted according to the value of their `priority` attribute, so that queues with a higher priority are considered before those with a lower priority. See section 2.2.5.3, “Prioritizing Execution Queues”, on page 23.

- Jobs are considered according to the priority of their queues. The scheduler runs all of the jobs that it can from the highest-priority queue before moving to the next queue, and so on. See section 4.8.4, “Examining Jobs Queue by Queue”, on page 129.

- Within each queue, jobs are considered in submission order.

- Starving jobs are given a special priority called `starving`. The default time required to become a starving job is 24 hours. See section 4.8.46, “Starving Jobs”, on page 279.

- Jobs in an express queue are placed in the `express_queue` preemption priority level. They are also placed in the `Express` execution priority class. The default priority for a queue to be an express queue is 150. See section 2.2.5.3.i, “Express Queues”, on page 24.

- Queued jobs are sorted according to their priority. Special jobs are all prioritized ahead of normal jobs, without regard to the queue in which they reside. The order for job priority for special jobs, highest first, is reservation jobs, jobs in express queues, starving jobs, suspended jobs. After this, the scheduler looks at normal jobs, queue by queue. All jobs in express queues, all starving jobs, and all suspended jobs are considered before the scheduler looks at the individual queues.

See section 4.8.16, “Calculating Job Execution Priority”, on page 166.

- The scheduler will preempt lower-priority jobs in order to run higher-priority jobs (preemptive_sched is True by default). By default, it has two levels of job priority, `express_queue`, and `normal_jobs`, where `express_queue` jobs can preempt...
Chapter 4  Scheduling

normal_jobs. This is set in the scheduler’s preempt_prio parameter.

When the scheduler chooses among jobs of the same priority for a job to preempt, it uses the default setting for preempt_sort, which is min_time_since_start, choosing jobs that have been running for the shortest time.

When the scheduler chooses how to preempt a job, it uses the default setting for its preempt_order parameter, which is SCR, meaning that first it will attempt suspension, then checkpointing, then if necessary requeueing.


• The scheduler will do its best to backfill smaller jobs around the job it has decided is the most important job. See section 4.8.3, “Using Backfilling”, on page 124.

• Primetime is 6:00 AM to 5:30 PM. Any holiday is considered non-primetime. Standard U.S. Federal holidays for the year are provided in the file PBS_HOME/sched_priv/holidays. These dates should be adjusted yearly to reflect your local holidays. See section 4.8.34, “Using Primetime and Holidays”, on page 241.

• The scheduler runs every 10 minutes unless a new job is submitted or a job finishes execution. See section 4.4.5, “The Scheduling Cycle”, on page 112.

4.3.8  Examples of Workload and Policy

• If you need to have high-priority jobs run soon, and nothing distinguishes the high-priority jobs from the rest:
  - Create advance reservations for the high-priority jobs, and have users submit those jobs to the reservations; see section 4.8.37, “Advance and Standing Reservations”, on page 249

• If you want to run jobs in submission order:
  - FIFO; see section 4.8.19, “FIFO Scheduling”, on page 180

• If you have low-priority jobs that should run only when other jobs don’t need the resources:
  - Set up an anti-express queue; see section 4.8.1, “Anti-Express Queues”, on page 120

• If you have a mix of jobs, and want to run big jobs first:
  - Sort jobs on a key, using ncpus as the key, to run big jobs first; see section 4.3.5, “Scheduling Jobs According to Size Etc.”, on page 91

• If you have a mix of jobs, and want to give big jobs high priority, but avoid having idle resources:
  - Sort jobs on a key, using ncpus as the key, to run big jobs first; see section 4.3.5.
Scheduling

“Scheduling Jobs According to Size Etc.”, on page 91
- Use backfilling; see section 4.8.3, “Using Backfilling”, on page 124

• If you want to have all users start about the same number of jobs:
  - Use round robin, and give each user their own queue; see section 4.8.38, “Round Robin Queue Selection”, on page 255

• If you want to always give each user access to a certain amount of a resource, but allow more if no one else is using it:
  - Use soft limits for the amount each user can use; see section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367
  - Use fairshare; see section 4.8.18, “Using Fairshare”, on page 170

• If you need to ensure that jobs run in certain hours on desktops:
  - Use cycle harvesting; see section 4.8.9, “Using Idle Workstation Cycle Harvesting”, on page 136
  - Use primetime & non-primetime for nighttime; see section 4.8.34, “Using Primetime and Holidays”, on page 241

• If you want to be sure a job will run:
  - Create an advance reservation; see section 4.8.37, “Advance and Standing Reservations”, on page 249

• If you have more than one complex, and you want to balance the workload across the complexes:
  - Use peer scheduling; see section 4.8.31, “Peer Scheduling”, on page 205

• If you have some jobs that should prefer to run on one set of vnodes, and other jobs that should prefer to run on another set of vnodes, but if the preferred vnodes are busy, a job can run on the non-preferred vnodes:
  - Use peer scheduling. Set up two complexes, give the pulling queues low priority, and use queue priority in the job sorting formula. See section 4.8.31, “Peer Scheduling”, on page 205
• If you have two (or more) sets of vnodes, and jobs should run on one set or the other, but not both. Additionally, jobs should not have to request where they run. If one set of vnodes is new, and one is old:
  - Use a routing queue and two execution queues. Associate each execution queue with one set of vnodes. Put the execution queue for the preferred set of vnodes first in the routing list, but put a limit on the number of queued jobs in the execution queues, so that both queues will fill up. Otherwise the routing queue will preferentially fill the first in its routing list. See section 2.2.6, “Routing Queues”, on page 24, and section 4.8.2, “Associating Vnodes with Queues”, on page 121

• If you need to apportion a single vnode or cluster according to ownership:
  - See section 4.3.4, “Allocating Resources by User, Project or Group”, on page 88

• If you have more than one high-priority queue, and at least one low-priority queue, and you want all jobs in high-priority queues to be considered as one group, and run in submission order:
  - Use the job sorting formula to sort jobs on queue priority:
    ```
    set server job_sort_formula = queue_priority
    ```
  - Give all queues whose jobs should be considered together the same priority
  - Set the by_queue scheduler attribute to True

• If you want to place jobs on the vnodes with the fewest CPUs first, saving bigger vnodes for larger jobs:
  - Sort vnodes so that those with fewer CPUs come first:
    ```
    node_sort_key: “ncpus LOW”
    ```

4.4 The Scheduler

The scheduler, pbs_sched, implements scheduling policy. The scheduler communicates with the MoMs to query the state of host-level resources and with the Server to learn about the availability of jobs to execute and the state of server-level resources. The interface to the server is through the same API as is used by the client commands. The scheduler communicates with the server with the privilege of a PBS Manager.
4.4.1 Configuring the Scheduler

4.4.1.1 Where the Scheduler Gets Its Information

The behavior of the scheduler is controlled by the information in the following options and attributes and files of parameters and settings:

The scheduler gets its information from the following sources:

- **PBS_HOME/sched_priv/resource_group**
  Contains the description of the fairshare tree. Created by you. Can be edited. Read on startup and HUP of scheduler.

- **PBS_HOME/sched_priv/usage**
  Contains the usage database. Do not edit. Instead, use the *pbsfs* command while the scheduler is stopped; see “pbsfs” on page 101 of the PBS Professional Reference Guide. Written every cycle and HUP. Read on startup. Cannot be altered while scheduler is running.

- **PBS_HOME/sched_priv/sched_config**
  Contains scheduler configuration options, also called scheduler parameters, e.g. *backfill*, *job_sort_key*. Read on startup and HUP. Can be edited. Each entry must be a single, unbroken line. Entries must be double-quoted if they contain whitespace.

- **PBS_HOME/sched_priv/dedicated_time**
  Contains definitions of dedicated time. Can be edited. Read on startup and HUP.

- **PBS_HOME/sched_priv/holidays**
  Contains definitions of holidays. Can be edited. Read on startup and HUP.

**Options to pbs_sched command**
Control some scheduler behavior. Set on invocation.

**Scheduler attributes**
Control some scheduler behavior. Can be set using *qmgr*. Read every scheduling cycle.

**Server attributes**
Several server attributes control scheduler behavior. Can be set using *qmgr*. The following table lists the server attributes that affect scheduling, along with a brief description. Read every scheduling cycle.

Some limit attributes are marked as “old”. These are incompatible with, and are replaced by, the new limit attributes described in section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367.
For a complete description of each attribute, see “Server Attributes” on page 316 of the PBS Professional Reference Guide.

Table 4-1: Server Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>backfill_depth</td>
<td>Modifies backfilling behavior. Sets the number of jobs that are to be backfilled around.</td>
</tr>
<tr>
<td>default_queue</td>
<td>Specifies queue for jobs that don’t request a queue</td>
</tr>
<tr>
<td>eligible_time_enable</td>
<td>Controls starving behavior.</td>
</tr>
<tr>
<td>est_start_time_freq</td>
<td>Interval at which PBS calculates estimated start times and vnodes for all jobs.</td>
</tr>
<tr>
<td>job_sort_formula</td>
<td>Formula for computing job priorities.</td>
</tr>
<tr>
<td>max_group_res</td>
<td>Old. The maximum amount of the specified resource that any single group may consume in this PBS complex.</td>
</tr>
<tr>
<td>max_group_res_soft</td>
<td>Old. The soft limit for the specified resource that any single group may consume in this complex.</td>
</tr>
<tr>
<td>max_group_run</td>
<td>Old. The maximum number of jobs owned by the users in one group allowed to be running within this complex at one time.</td>
</tr>
<tr>
<td>max_group_run_soft</td>
<td>Old. The maximum number of jobs owned by the users in one group allowed to be running in this complex at one time.</td>
</tr>
</tbody>
</table>
Table 4-1: Server Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_queued</td>
<td>The maximum number of jobs allowed to be queued in or running in the complex. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_queued_res.&lt;resource&gt;</td>
<td>The maximum amount of the specified resource allowed to be allocated to jobs queued in or running in the complex. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_run</td>
<td>The maximum number of jobs allowed to be running in the complex. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_run_res.&lt;resource&gt;</td>
<td>The maximum amount of the specified resource allowed to be allocated to jobs running in the complex. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_run_res_soft.&lt;resource&gt;</td>
<td>Soft limit on the amount of the specified resource allowed to be allocated to jobs running in the complex. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_run_soft</td>
<td>Soft limit on the number of jobs allowed to be running in the complex. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_running</td>
<td>Old. The maximum number of jobs allowed to be selected for execution at any given time, from all possible jobs.</td>
</tr>
</tbody>
</table>
### Table 4-1: Server Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_user_res</td>
<td>Old. The maximum amount within this complex that any single user may consume of the specified resource.</td>
</tr>
<tr>
<td>max_user_res_soft</td>
<td>Old. The soft limit on the amount of the specified resource that any single user may consume within a complex.</td>
</tr>
<tr>
<td>max_user_run</td>
<td>Old. The maximum number of jobs owned by a single user allowed to be running within the complex at one time.</td>
</tr>
<tr>
<td>max_user_run_soft</td>
<td>Old. The soft limit on the number of jobs owned by a single user that are allowed to be running within this complex at one time.</td>
</tr>
<tr>
<td>node_fail_requeue</td>
<td>Controls whether running jobs are automatically requeued or are deleted when the primary execution vnode fails. Number of seconds to wait after losing contact with Mother Superior before requeueing or deleting jobs.</td>
</tr>
<tr>
<td>node_group_enable</td>
<td>Specifies whether node grouping is enabled.</td>
</tr>
<tr>
<td>node_group_key</td>
<td>Specifies the resource to use for node grouping.</td>
</tr>
<tr>
<td>resources_available</td>
<td>The list of available resources and their values defined on the server.</td>
</tr>
</tbody>
</table>
Several vnode attributes control scheduler behavior. Can be set using `qmgr`. The following table lists the vnode attributes that affect scheduling, along with a brief description. Read every scheduling cycle. For a complete description of each attribute, see “Vnode Attributes” on page 366 of the PBS Professional Reference Guide.

**Table 4-2: Vnode Attributes Involved in Scheduling**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_aoe</td>
<td>This attribute identifies the AOE currently instantiated on this vnode.</td>
</tr>
<tr>
<td>max_group_run</td>
<td>The maximum number of jobs owned by any users in a single group allowed to run on this vnode at one time.</td>
</tr>
<tr>
<td>max_running</td>
<td>The maximum number of jobs allowed to be run on this vnode at any given time.</td>
</tr>
</tbody>
</table>
Several queue attributes control scheduler behavior. These are attributes such as `node_group_key`. Can be set using `qmgr`. The following table lists the queue attributes that affect scheduling, along with a brief description. Read every

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_user_run</td>
<td>The maximum number of jobs owned by a single user allowed to run on this vnode at one time.</td>
</tr>
<tr>
<td>no_multinode_jobs</td>
<td>Controls whether jobs which request more than one chunk are allowed to execute on this vnode.</td>
</tr>
<tr>
<td>priority</td>
<td>The priority of this vnode compared with other vnodes.</td>
</tr>
<tr>
<td>provision_enable</td>
<td>Controls whether this vnode can be provisioned.</td>
</tr>
<tr>
<td>queue</td>
<td>The queue with which this vnode is associated.</td>
</tr>
<tr>
<td>resources_available</td>
<td>The list of resources and the amounts available on this vnode</td>
</tr>
<tr>
<td>sharing</td>
<td>Specifies whether more than one job at a time can use the resources of the vnode or the vnode’s host.</td>
</tr>
<tr>
<td>state</td>
<td>Shows or sets the state of the vnode.</td>
</tr>
<tr>
<td>pcpus</td>
<td>The number of physical CPUs on the vnode.</td>
</tr>
<tr>
<td>resources_assigned</td>
<td>The total amount of each resource allocated to jobs and started reservations running on this vnode.</td>
</tr>
</tbody>
</table>
scheduling cycle. For a complete description of each attribute, see “Queue Attributes” on page 354 of the PBS Professional Reference Guide.

Table 4-3: Queue Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Specifies whether this queue accepts new jobs.</td>
</tr>
<tr>
<td>from_route_only</td>
<td>Specifies whether this queue accepts jobs only from routing queues, or from both execution and routing queues.</td>
</tr>
<tr>
<td>max_array_size</td>
<td>The maximum number of subjobs that are allowed in an array job.</td>
</tr>
<tr>
<td>max_group_res</td>
<td>Old. The maximum amount of the specified resource that any single group may consume in this queue.</td>
</tr>
<tr>
<td>max_group_res_soft</td>
<td>Old. The soft limit for the specified resource that any single group may consume in this queue.</td>
</tr>
<tr>
<td>max_group_run</td>
<td>Old. The maximum number of jobs owned by the users in one group allowed to be running within this queue at one time.</td>
</tr>
<tr>
<td>max_group_run_soft</td>
<td>Old. The maximum number of jobs owned by the users in one group allowed to be running in this queue at one time.</td>
</tr>
<tr>
<td>max_queuable</td>
<td>Old. The maximum number of jobs allowed to reside in the queue at any given time.</td>
</tr>
<tr>
<td>max_queued</td>
<td>The maximum number of jobs allowed to be queued in or running from the queue. Can be specified for users, groups, or all.</td>
</tr>
</tbody>
</table>
Table 4-3: Queue Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_queued_res.&lt;resource&gt;</td>
<td>The maximum amount of the specified resource allowed to be allocated to jobs queued in or running from the queue. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_run</td>
<td>The maximum number of jobs allowed to be running from the queue. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_run_res.&lt;resource&gt;</td>
<td>The maximum amount of the specified resource allowed to be allocated to jobs running from the queue. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_run_res_soft.&lt;resource&gt;</td>
<td>Soft limit on the amount of the specified resource allowed to be allocated to jobs running from the queue. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_run_soft</td>
<td>Soft limit on the number of jobs allowed to be running from the queue. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_running</td>
<td>Old. The maximum number of jobs allowed to be selected for execution at any given time, from all possible jobs.</td>
</tr>
<tr>
<td>max_user_res</td>
<td>Old. The maximum amount of the specified resource that the jobs of any single user may consume.</td>
</tr>
<tr>
<td>max_user_res_soft</td>
<td>Old. The soft limit on the amount of the specified resource that any single user may consume in this queue.</td>
</tr>
</tbody>
</table>
### Table 4-3: Queue Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_user_run</td>
<td>Old. The maximum number of jobs owned by a single user allowed to be running from the queue at one time.</td>
</tr>
<tr>
<td>max_user_run_soft</td>
<td>Old. The soft limit on the number of jobs owned by a single user that are allowed to be running from this queue at one time.</td>
</tr>
<tr>
<td>node_group_key</td>
<td>Specifies the resource to use for node grouping.</td>
</tr>
<tr>
<td>Priority</td>
<td>The priority of this queue compared to other queues of the same type in this PBS complex.</td>
</tr>
<tr>
<td>resources_assigned</td>
<td>The total of each type of resource allocated to jobs running in this queue.</td>
</tr>
<tr>
<td>resources_available</td>
<td>The list of available resources and their values defined on the queue.</td>
</tr>
<tr>
<td>resources_max</td>
<td>The maximum amount of each resource that can be requested by any single job in this queue.</td>
</tr>
<tr>
<td>resources_min</td>
<td>The minimum amount of each resource that can be requested by a single job in this queue.</td>
</tr>
<tr>
<td>route_destinations</td>
<td>The list of destinations to which jobs may be routed.</td>
</tr>
<tr>
<td>route_held_jobs</td>
<td>Specifies whether jobs in the held state can be routed from this queue.</td>
</tr>
<tr>
<td>route_lifetime</td>
<td>The maximum time a job is allowed to reside in a routing queue. If a job cannot be routed in this amount of time, the job is aborted.</td>
</tr>
</tbody>
</table>
Chapter 4  

**Scheduling**

### Table 4-3: Queue Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>route_retry_time</td>
<td>Time delay between routing retries. Typically used when the network between servers is down.</td>
</tr>
<tr>
<td>route_waiting_jobs</td>
<td>Specifies whether jobs whose <code>execution_time</code> attribute value is in the future can be routed from this queue.</td>
</tr>
<tr>
<td>started</td>
<td>Specifies whether jobs in this queue can be scheduled for execution.</td>
</tr>
<tr>
<td>state_count</td>
<td>The number of jobs in each state currently residing in this queue.</td>
</tr>
</tbody>
</table>

**List of jobs and server-level resources queried from Server**  
Cannot be altered. Read every scheduling cycle.

**List of host-level resources queried from MoMs**  
Cannot be altered. Read every scheduling cycle.

### 4.4.1.2 Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

### 4.4.1.3 Reference Copies of Files

PBS is installed with a reference copy of the current year’s holidays file, in `PBS_EXEC/etc/pbs_holidays`.

### 4.4.2 Making the Scheduler Read its Configuration

If you change the scheduler’s configuration file, the scheduler must re-read it for the changes to take effect. To get the scheduler to re-read its configuration information, without stopping the scheduler, you can HUP the scheduler:

```
  kill -HUP <scheduler PID>
```
4.4.3 Scheduling on Resources

The scheduler honors all resources listed in the `resources:` line in `PBS_HOME/sched_priv/sched_config`. If this line is not present, the scheduler honors all resources, built-in and custom. It is more efficient to list just the resources that you want the scheduler to schedule on.

4.4.4 Starting, Stopping, and Restarting the Scheduler

4.4.4.1 When and How to Start the Scheduler

During normal operation, startup of the scheduler is handled automatically. The PBS daemons are started automatically at bootup by the PBS start/stop script. During failover, the secondary server automatically tries to use the primary scheduler, and if it cannot, it starts its own scheduler.

To start the scheduler by hand:

```
PBS_EXEC/sbin/pbs_sched [options]
```

See “pbs_sched” on page 87 of the PBS Professional Reference Guide.

4.4.4.2 When and How to Stop the Scheduler

You must stop the scheduler for the following operations:

- Using the `pbsfs` command; see “pbsfs” on page 101 of the PBS Professional Reference Guide.
- Upgrading PBS Professional; see “Upgrading PBS Professional” on page 111 in the PBS Professional Installation & Upgrade Guide.

The scheduler traps signals during the scheduling cycle. You can kill the scheduler at the end of the cycle, or if necessary, immediately. The scheduler does not write the fairshare usage file when it is killed with -9, but it does write the file when it is killed without -9.

You must be root on the scheduler’s host.

To stop the scheduler at the end of a cycle:

```
kill <scheduler PID>
```

To stop the scheduler immediately:

```
kill -9 <scheduler PID>
```
4.4.4.3 When and How to Restart the Scheduler

Under most circumstances, when you restart the scheduler, you do not need to specify any options to the `pbs_sched` command. See “pbs_sched” on page 87 of the PBS Professional Reference Guide. Start the scheduler this way:

```
PBS_EXEC/sbin/pbs_sched [options]
```

The PBS start/stop script is not recommended for restarting the scheduler after stopping it, because the script causes any running jobs to be killed.

4.4.5 The Scheduling Cycle

The scheduler runs in a loop. Inside each loop, it starts up, performs all of its work, and then stops. The scheduling cycle is triggered by a timer and by several possible events.

When there are no events to trigger the scheduling cycle, it is started by a timer. The time between starts is set in the server’s `scheduler_iteration` server attribute. The default value is 10 minutes.

The maximum duration of the cycle is set in the scheduler’s `sched_cycle_length` attribute. The scheduler will terminate its cycle if the duration of the cycle exceeds the value of the attribute. The default value for the length of the scheduling cycle is 20 minutes. The scheduler does not include the time it takes to query dynamic resources in its cycle measurement.

4.4.5.1 Triggers for Scheduling Cycle

The scheduler starts when the following happen:

- The specified amount of time has passed since the previous start
- A job is submitted
- A job finishes execution.
- A new reservation is created
- A reservation starts
- Scheduling is enabled
- The server comes up
- A job is qrun
- A queue is started
- A job is moved to a local queue
- Eligible wait time for jobs is enabled
- A reservation is re-confirmed after being degraded
4.4.5.1 Logging Scheduling Trigger

The server triggers scheduler cycles. The reason for triggering a scheduling cycle is logged by the server. See section 12.4.5.2, “Scheduler Commands”, on page 872.

4.4.5.2 Actions During Scheduling Cycle

The following is a list of the scheduler’s actions during a scheduling cycle. The list is not in any special order.

- The scheduler gets the state of the world:
  - The scheduler queries the server for the following:
    - Status of jobs in queues
    - All global server, queue, and host-level resources
    - Server, queue, vnode, and scheduler attribute settings
    - Reservations
  - The scheduler runs dynamic server resource queries for resources listed in “server_dyn_res” line in sched_config
  - The scheduler runs dynamic host-level resource queries for resources listed in “mom_resources” line in sched_config
- The scheduler logs a message at the beginning of each scheduling cycle saying whether it is primetime or not, and when this period of primetime or non-primetime will end. The message is at log event class 0x0100. The message is of this form:
  “It is primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS”
  or
  “It is non-primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS”
- Given scheduling policy, available jobs and resources, and scheduling cycle length, the scheduler examines as many jobs as it can, and runs as many jobs as it can.

4.4.6 How Available Consumable Resources are Counted

When the scheduler checks for available consumable resources, it uses the following calculation:

\[ \text{resources\_available.<resource>} - \text{total resources assigned for this resource} \]

\text{total resources assigned} is the total amount of resources\_assigned.<resource> for all other running jobs and, at the server and vnodes, for started reservations.
For example, if the scheduler is calculating available memory, and two other jobs are running, each with 2GB of memory assigned, and resources_available.mem is 8GB, the scheduler figures that it has 4GB to work with.

### 4.5 Using Queues in Scheduling

A queue is a PBS mechanism for holding jobs. PBS has queue-based tools for handling jobs; for example, you can set queue-based limits on resource usage by jobs. PBS uses queues for a variety of purposes. Before reading this section, please familiarize yourself with the mechanics of creating and configuring queues, by reading section 2.2, “Queues”, on page 18.

Queues are used in the following ways:

- Holding submitted jobs
- Prioritizing jobs and ordering job selection:
  - PBS provides tools for selecting jobs according to the queue they are in; see section 4.2.5.3, “Using Queue-based Tools to Prioritize Jobs”, on page 68
  - Queue priority can be used in calculating job priority; see section 4.8.36, “Queue Priority”, on page 248
- Providing tools for managing time slots
  - Reservations: you can reserve specific resources for defined time slots. Queues are used for advance and standing reservations; see section 4.8.37, “Advance and Standing Reservations”, on page 249, and "Reserving Resources Ahead of Time", on page 185 of the PBS Professional User’s Guide
  - Dedicated time; see section 4.8.10, “Dedicated Time”, on page 153
  - Primetime and holidays; see section 4.8.34, “Using Primetime and Holidays”, on page 241
- Routing jobs: Many ways to route jobs are listed in section 4.8.39, “Routing Jobs”, on page 256
- Providing tools for managing resources
  - Managing resource usage by users; see section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367
  - Managing resource usage by jobs; see section 5.15.3, “Placing Resource Limits on Jobs”, on page 391
  - Setting resource and job limits used for preemption: you can specify how much of a resource or how many jobs a user or group can use before their jobs are eligible to be preempted. See section 5.15.1.4, “Hard and Soft Limits”, on page 371 and section
4.6 Scheduling Restrictions and Caveats

4.6.1 One Policy Per PBS Complex

The scheduler runs a single scheduling policy, and applies it to the entire PBS complex. You cannot have two different scheduling policies on two different queues or partitions.

4.6.2 Jobs that Cannot Run on Current Resources

The scheduler checks to see whether each job could possibly run now, counting resources as if there were no other jobs, and all current resources could be used by this job. The scheduler counts resources only from those vnodes that are on line. If a vnode is marked offline, its resources are not counted.

A job that cannot run on current resources will sit in the queue until it becomes the most deserving job. Whenever this job is considered for being run, and backfilling is being used, the following error message is printed in the scheduler’s log file:

“resource request is impossible to solve: job will never run”

The scheduler then examines the next job in line to be the most deserving job.

The scheduler determines whether a job cannot run on current resources only when backfilling is used. If backfilling is turned off, then the scheduler won't determine whether or not a job has requested more than can be supplied by current resources. It decides only that it can't run now.

4.6.3 Resources Not Controlled by PBS

When the scheduler runs each cycle, it gets the state of its world, including dynamic resources outside of the control of PBS. If non-PBS processes are running on the vnodes PBS uses, it is possible that another process will use enough of a dynamic resource such as scratch space to prevent a PBS job that requested that resource from running.
Chapter 4  

Scheduling

4.6.4  No Pinning of Processes to Cores

PBS does not pin processes to cores. This can be accomplished in the job launch script using, for example, `taskset` or `dplace`.

4.7  Errors and Logging

4.7.1  Logfile for scheduler

You can set the scheduler’s logging to record different kinds of events. See section 12.4.4.1.iii, “Specifying Scheduler Log Events”, on page 871.

The server triggers scheduler cycles. The reason for triggering a scheduling cycle is logged by the server. See section 12.4.5.2, “Scheduler Commands”, on page 872.

4.8  Scheduling Tools

In this section (all of section 4.8, “Scheduling Tools”, on page 116, and its subsections), we describe each scheduling tool, including how to configure it.

The following table lists PBS scheduling tools, with links to descriptions:

<table>
<thead>
<tr>
<th>Scheduling Tool</th>
<th>Incompatible Tools</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anti-express queue</td>
<td>soft queue limits</td>
<td>See section 4.8.1, “Anti-Express Queues”, on page 120</td>
</tr>
<tr>
<td>Associating vnodes with queues</td>
<td></td>
<td>See section 4.8.2, “Associating Vnodes with Queues”, on page 121</td>
</tr>
<tr>
<td>Backfilling</td>
<td>fairshare or preemption w/ backfilling+strict ordering</td>
<td>See section 4.8.3, “Using Backfilling”, on page 124</td>
</tr>
<tr>
<td>Examining jobs queue-by-queue</td>
<td>round robin, queues as fairshare entities</td>
<td>See section 4.8.4, “Examining Jobs Queue by Queue”, on page 129</td>
</tr>
<tr>
<td>Checkpointing</td>
<td></td>
<td>See section 4.8.5, “Checkpointing”, on page 131</td>
</tr>
</tbody>
</table>
### Table 4-4: List of Scheduling Tools

<table>
<thead>
<tr>
<th>Scheduling Tool</th>
<th>Incompatible Tools</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organizing job chunks</td>
<td></td>
<td>See section 4.8.6, “Organizing Job Chunks”, on page 131</td>
</tr>
<tr>
<td><code>cron</code> jobs, Windows Task Scheduler</td>
<td></td>
<td>See section 4.8.7, “<code>cron</code> Jobs, or the Windows Task Scheduler”, on page 132</td>
</tr>
<tr>
<td>Custom resources</td>
<td></td>
<td>See section 4.8.8, “Using Custom and Default Resources”, on page 133</td>
</tr>
<tr>
<td>Cycle harvesting, reservations</td>
<td></td>
<td>See section 4.8.9, “Using Idle Workstation Cycle Harvesting”, on page 136</td>
</tr>
<tr>
<td>Dedicated time</td>
<td></td>
<td>See section 4.8.10, “Dedicated Time”, on page 153</td>
</tr>
<tr>
<td>Default resources</td>
<td></td>
<td>See section 4.8.8, “Using Custom and Default Resources”, on page 133</td>
</tr>
<tr>
<td>Dependencies</td>
<td></td>
<td>See section 4.8.11, “Dependencies”, on page 154</td>
</tr>
<tr>
<td>Dynamic resources (server &amp; host)</td>
<td></td>
<td>See section 4.8.12, “Dynamic Resources”, on page 154</td>
</tr>
<tr>
<td>Eligible wait time for jobs</td>
<td></td>
<td>See section 4.8.13, “Eligible Wait Time for Jobs”, on page 155</td>
</tr>
<tr>
<td>Entity shares (was strict priority)</td>
<td>formula, fairshare, FIFO</td>
<td>See section 4.8.14, “Sorting Jobs by Entity Shares (Was Strict Priority)”, on page 160</td>
</tr>
<tr>
<td>Calculating job execution priority</td>
<td></td>
<td>See section 4.8.16, “Calculating Job Execution Priority”, on page 166</td>
</tr>
</tbody>
</table>

---

PBS Professional 12.2 Administrator’s Guide 117
## Table 4-4: List of Scheduling Tools

<table>
<thead>
<tr>
<th>Scheduling Tool</th>
<th>Incompatible Tools</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Express queues</td>
<td>See section 4.8.17, “Express Queues”, on page 169</td>
<td></td>
</tr>
<tr>
<td>Fairshare</td>
<td>formula, starving, strict ordering, using the fair_share_perc option to job_sort_key</td>
<td>See section 4.8.18, “Using Fairshare”, on page 170</td>
</tr>
<tr>
<td>FIFO</td>
<td>See section 4.8.19, “FIFO Scheduling”, on page 180</td>
<td></td>
</tr>
<tr>
<td>Formula</td>
<td>job_sort_key, fairshare</td>
<td>See section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182</td>
</tr>
<tr>
<td>Gating jobs at server or queue</td>
<td>See section 4.8.21, “Gating Jobs at Server or Queue”, on page 190</td>
<td></td>
</tr>
<tr>
<td>Managing application licenses</td>
<td>See section 4.8.22, “Managing Application Licenses”, on page 191</td>
<td></td>
</tr>
<tr>
<td>Limits on per-job resource usage</td>
<td>See section 4.8.23, “Limits on Per-job Resource Usage”, on page 192</td>
<td></td>
</tr>
<tr>
<td>Limits on project, user, and group jobs</td>
<td>See section 4.8.24, “Limits on Project, User, and Group Jobs”, on page 192</td>
<td></td>
</tr>
<tr>
<td>Limits on project, user, and group resource usage</td>
<td>See section 4.8.25, “Limits on Project, User, and Group Resource Usage”, on page 192</td>
<td></td>
</tr>
<tr>
<td>Limits on jobs at vnodes</td>
<td>See section 4.8.26, “Limits on Jobs at Vnodes”, on page 193</td>
<td></td>
</tr>
<tr>
<td>Load balancing</td>
<td>node_sort_key using unused or assigned options,</td>
<td>See section 4.8.27, “Using Load Balancing”, on page 193</td>
</tr>
</tbody>
</table>
### Table 4-4: List of Scheduling Tools

<table>
<thead>
<tr>
<th>Scheduling Tool</th>
<th>Incompatible Tools</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matching jobs to resources</td>
<td></td>
<td>See section 4.8.28, “Matching Jobs to Resources”, on page 197</td>
</tr>
<tr>
<td>Node grouping</td>
<td></td>
<td>See section 4.8.29, “Node Grouping”, on page 200</td>
</tr>
<tr>
<td>Overrides</td>
<td></td>
<td>See section 4.8.30, “Overrides”, on page 201</td>
</tr>
<tr>
<td>Peer scheduling</td>
<td></td>
<td>See section 4.8.31, “Peer Scheduling”, on page 205</td>
</tr>
<tr>
<td>Placement sets</td>
<td></td>
<td>See section 4.8.32, “Placement Sets”, on page 211</td>
</tr>
<tr>
<td>Preemption</td>
<td></td>
<td>See section 4.8.33, “Using Preemption”, on page 228</td>
</tr>
<tr>
<td>Primetime and holidays</td>
<td></td>
<td>See section 4.8.34, “Using Primetime and Holidays”, on page 241</td>
</tr>
<tr>
<td>Provisioning</td>
<td></td>
<td>See section 4.8.35, “Provisioning”, on page 247</td>
</tr>
<tr>
<td>Queue priority</td>
<td></td>
<td>See section 4.8.36, “Queue Priority”, on page 248</td>
</tr>
<tr>
<td>Advance and standing reservations</td>
<td>cycle harvesting</td>
<td>See section 4.8.37, “Advance and Standing Reservations”, on page 249</td>
</tr>
<tr>
<td>Round robin queue examination</td>
<td>by_queue</td>
<td>See section 4.8.38, “Round Robin Queue Selection”, on page 255</td>
</tr>
<tr>
<td>Routing jobs</td>
<td></td>
<td>See section 4.8.39, “Routing Jobs”, on page 256</td>
</tr>
<tr>
<td>Shared or exclusive vnodes and hosts</td>
<td></td>
<td>See section 4.8.40, “Shared vs. Exclusive Use of Resources by Jobs”, on page 261</td>
</tr>
</tbody>
</table>
4.8.1 Anti-Express Queues

An anti-express queue is a preemptable low-priority queue, designed for jobs that should run only when no other jobs need the resources. These jobs are preempted if any other job needs the resources. An anti-express queue has the lowest priority of all queues in the complex. Jobs in this queue have a soft limit of zero, so that any job running from this queue is over its queue soft limit.

4.8.1.1 Configuring Anti-express Queues

To configure an anti-express queue, do the following:

- Create an execution queue called lowprio:
  ```
  Qmgr: create queue lowprio
  Qmgr: set queue lowprio queue_type=e
  Qmgr: set queue lowprio started=true
  Qmgr: set queue lowprio enabled=true
  ```

- By default, all new queues have a priority of zero. Make sure all queues have a value set for priority, and that lowprio has the lowest priority:
  ```
  Qmgr: set queue workq priority=10
  ```

- Set the soft limit on the number of jobs that can run from that queue to zero for all users:
  ```
  Qmgr: set queue lowprio max_run_soft = "[u:PBS_GENERIC=0]"
  ```

- Make sure that jobs over their queue soft limits have lower preemption priority than normal jobs. Edit PBS_HOME/sched_priv/sched_config, and do the following:
  - Add “queue_softlimits” after “normal_jobs”. For example:
    ```
    preempt_prio: "express_queue, normal_jobs, queue_softlimits"
    ```
  - Sort queues:
    ```
    sort_queues: True  ALL
    ```
  - Use preemption:
    ```
    preemptive_sched True  ALL
    ```

The sort_queues parameter is deprecated as of 12.2.

4.8.1.2 Anti-express Queue Caveats

If you use soft limits on the number of jobs that users can run at other queues, jobs that are over their soft limits at other queues will also have the lowest preemption priority.

4.8.2 Associating Vnodes with Queues

You can associate each vnode with one or more queues. When a vnode is associated with a queue, that means it accepts jobs from that queue only. There are two arrangements:

- One or more vnodes associate with one queue
- One or more vnodes associate with multiple queues

These two arrangements require different methods of configuration.
You do not need to associate vnodes with queues in order to have jobs run on the vnodes that have the right application, as long as the application is a resource that can be requested by jobs. See section 5.14.7, “Supplying Application Licenses”, on page 346.

4.8.2.1 Associating Vnodes With One Queue

You can associate one or more vnodes with a queue, using the vnode’s queue attribute. Using this method, each vnode can be associated with at most one queue. Each queue can be associated with more than one vnode. If you associate a queue and one or more vnodes using this method, any jobs in the queue can run only on the associated vnodes, and the only jobs that can run on the vnodes are the ones in the queue.

To associate a vnode with a queue, set the vnode’s queue attribute to the name of the queue you want. For example, to associate the vnode named Vnode1 with the queue named Queue1:

```
Qmgr: set node Vnode1 queue=Queue1
```

4.8.2.2 Associating Vnodes With Multiple Queues

You can use custom host-level resources to associate one or more vnodes with more than one queue. The scheduler will use the resources for scheduling just as it does with any resource.

In order to map a vnode to more than one queue, you must define a new host-level string array custom resource. This string array holds a string that has the same value for the queue and vnode you wish to associate. The mechanism of association is that a job that lands in the queue inherits that value for the resource, and then the job can run only on vnodes having a matching value for the resource. You can associate more than one queue with a vnode by setting the resource to the same value at each queue.

In some cases, you can use the same resource to route jobs and to associate vnodes with queues. For the method described here, you use host-level resources to associate vnodes with queues. The rules for which resources can be used for routing are given in section 2.2.6.4.iii, “Resources Used for Routing and Admittance”, on page 27. How jobs inherit resources is described in section 5.9.4, “Allocating Default Resources to Jobs”, on page 309.
4.8.2.2.i  Procedure to Associate Vnodes with Multiple Queues

To associate one or more vnodes with one or more queues, do the following:

1. Define the new host-level resource. Add the new resource to $PBS_HOME/server_priv/resourcedef:
   
   `<new resource> type=string_array flag=h`


3. Instruct the scheduler to honor the resource. Add the new resource to $PBS_HOME/sched_priv/sched_config:
   
   `resources: "ncpus, mem, arch, host, vnode, <new resource>"`

4. HUP the scheduler:
   
   `kill -HUP <scheduler PID>`

5. Set each queue’s `default_chunk` for the new resource to the value you are using to associate it with vnodes:
   
   `Qmgr: set queue <queue name> default_chunk.<new resource> = <value>`
   
   For example, if one queue is “MathQ” and one queue is “SpareQ”, and the new resource is “Qlist”, and you want to associate a set of vnodes and queues based on ownership by the math department, you can make the queue resource value be “math”:
   
   `Qmgr: set queue MathQ default_chunk.Qlist = math`
   `Qmgr: set queue SpareQ default_chunk.Qlist = math`

6. Set the value for the new resource at each vnode:
   
   `Qmgr: set node <vnode name> resources_available.<new resource> = <associating value>`
   
   For example, to have the vnode named “Vnode1” associated with the queues owned by the math department:
   
   `Qmgr: set node Vnode1 resources_available.Qlist = math`

4.8.2.2.ii  Example of Associating Multiple Vnodes with Multiple Queues

Now, as an example, assume you have 2 queues: “PhysicsQ” and “ChemQ”, and you have 3 vnodes: vn[1], vn[2], and vn[3]. You want Physics jobs to run on vn[1] and vn[2], and you want Chem jobs to run on vn[2] and vn[3]. Each department gets exclusive use of one vnode, but both must share a vnode.
4.8.3 Using Backfilling

Backfilling means fitting smaller jobs around the higher-priority jobs that the scheduler is going to run next, in such a way that the higher-priority jobs are not delayed.

Backfilling allows you to keep resources from becoming idle when the top job cannot run.

Backfilling is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.
4.8.3.1 Terminology

**Top job**
Job which has the highest execution priority according to scheduling policy. The scheduler plans resources and start time for this job first. The scheduler does not allow other jobs to delay the start of a top job, whether or not backfilling is being used.

**Filler job**
Smaller job that fits around top jobs. Running a filler job does not change the start time or resources for a top job. This job runs next only when backfilling is being used (meaning that a top job cannot start next because insufficient resources are available for the top job, but whatever is available is enough for the filler job).

4.8.3.2 How Backfilling Works

The scheduler makes a list of jobs to run in order of priority. This list is composed according to execution priority described in section 4.8.16, “Calculating Job Execution Priority”, on page 166.

If you use backfilling, the scheduler looks for smaller jobs that can fit into the usage gaps around the highest-priority jobs in the list. The scheduler looks in the prioritized list of jobs and chooses the highest-priority smaller jobs that fit. Filler jobs are run only if they will not delay the start time of top jobs.

The scheduler re-creates the list of highest-priority jobs at every scheduling cycle.

You can use shrink-to-fit jobs to backfill into otherwise unusable time slots. PBS checks whether a shrink-to-fit job could shrink into the available slot, and if it can, runs it. See section 4.8.41, “Using Shrink-to-fit Jobs”, on page 263.

Backfilling is useful in the following circumstances:

- When the **strict_ordering** scheduler parameter is turned on, filler jobs are fitted around higher-priority jobs. Without backfilling, no job runs if the top job cannot run. See section 4.8.47, “Using Strict Ordering”, on page 282

- When the **help_starving_jobs** scheduler parameter is turned on, filler jobs are fitted around starving jobs. See section 4.8.46, “Starving Jobs”, on page 279
4.8.3.3 Backfilling Around N Jobs

You can configure the number of top jobs that PBS backfills around by setting the value of the backfill_depth server attribute. For example, if you set backfill_depth to 3, PBS backfills around the top 3 jobs. See “Server Attributes” on page 316 of the PBS Professional Reference Guide. Setting the backfill_depth parameter is effective only when backfill is set to True.

4.8.3.4 Configuring Backfilling

To configure backfilling, do the following:
1. Choose how many jobs to backfill around. If you want to backfill around more than 1 job, set the backfill_depth server attribute to the desired number. The default is 1. Set this parameter to less than 100.
2. Choose whether to use backfilling during primetime, non-primetime, or all of the time. If you want separate primetime and non-primetime behavior, specify the backfill parameter twice, once for each. The default is all.
3. Make sure that the backfill scheduler parameter is True for the time you want it. The default is True. For example:
   - backfill True prime
   - backfill False non_prime
4. Make sure that jobs request walltime by making them inherit a walltime resource if they don’t explicitly request it. For options, see section 4.8.3.8.i, “Ensure Jobs Are Eligible for Backfilling”, on page 127.

4.8.3.5 Backfilling and Strict Ordering

When you use strict ordering, the scheduler runs jobs in exactly the order of their priority. If backfilling is turned off and the top job cannot run, no job is able to run. Backfilling can prevent resources from standing idle while the top job waits for its resources to become available. See section 4.8.47, “Using Strict Ordering”, on page 282.

4.8.3.6 Backfilling and Starving Jobs

When you take starving jobs into consideration, by setting the help_starving_jobs scheduler parameter to True, starving jobs can become the top jobs. They can continue to wait for resources once they are the top job, blocking other jobs from running. Backfilling can prevent resources from standing idle while the top job waits for its resources to become available. See section 4.8.46, “Starving Jobs”, on page 279.
4.8.3.7 Attributes and Parameters Affecting Backfilling

backfill
Scheduler parameter. Controls whether or not PBS uses backfilling. Scheduler will backfill when either strict_ordering is True or help_starving_jobs is True. See “backfill” on page 282 of the PBS Professional Reference Guide.

backfill_depth
Server attribute. Modifies backfilling behavior. Sets the number of jobs that are to be backfilled around. See “Server Attributes” on page 316 of the PBS Professional Reference Guide.

The configuration parameters backfill_prime and prime_exempt_anytime_queues do not relate to backfilling. They control the time boundaries of regular jobs with respect to primetime and non-primetime. See section 4.8.34, “Using Primetime and Holidays”, on page 241.

4.8.3.8 Backfilling Recommendations and Caveats

4.8.3.8.i Ensure Jobs Are Eligible for Backfilling
When calculating backfilling, PBS treats a job that has no walltime specified as if its walltime is eternity. The scheduler will never use one of these jobs as a filler job. You can avoid this by ensuring that each job has a realistic walltime, by using the following methods:

- At qsub time via a hook
- By setting the queue’s resources_default.walltime attribute
- By setting the server’s resources_default.walltime attribute
- At qsub time via the server’s default_qsub_arguments

4.8.3.8.ii Number of Jobs to Backfill Around
The more jobs being backfilled around, the longer the scheduling cycle takes.

4.8.3.8.iii Dynamic Resources and Backfilling
Using dynamic resources and backfilling may result in some jobs not being run even though resources are available. This may happen when a job requesting a dynamic resource is selected as the top job. The scheduler must estimate when resources will become available, but it can only query for resources available at the time of the query, not resources already in use, so it will not be able to predict when resources in use become available. Therefore the scheduler won’t be able to schedule the job. In addition, since dynamic resources are outside of the control of PBS, they may be consumed between the time the scheduler queries for the resource and the time it starts a job.
4.8.3.8.iv  Avoid Using Strict Ordering, Backfilling, and Fairshare

It is inadvisable to use strict ordering and backfilling with fairshare.

The results may be non-intuitive. Fairshare will cause relative job priorities to change with each scheduling cycle. It is possible that a job from the same entity or group will be chosen as the small job. The usage from these small jobs will lower the priority of the most deserving job.

For example, if a user has a large job that is the most deserving but cannot run, smaller jobs owned by that user will chew up the user's usage, and prevent the large job from ever being likely to run. Also, if the small jobs are owned by a user in one area of the fairshare tree, no large jobs owned by anyone else in that section of the fairshare tree are likely to be able to run.

4.8.3.8.v  Avoid Using Preemption, Strict Ordering, and Backfilling

Using preemption with strict ordering and backfilling may change the top job.

4.8.3.8.vi  Warning About Backfilling and Provisioning

The scheduler will not run a job requesting an AOE on a vnode that has a top job scheduled on it in the future.

The scheduler will not use a job requesting an AOE as a top job.

4.8.3.8.vii  Jobs That Cannot Run on Current Resources

The scheduler checks to see whether each job could possibly run now, counting resources as if there were no other jobs, and all current resources could be used by this job. The scheduler counts resources only from those vnodes that are on line. If a vnode is marked offline, its resources are not counted.

When the scheduler is using backfilling, if the top job cannot run on current resources, the following error message is printed in the scheduler’s log file:

"resource request is impossible to solve: job will never run"

The scheduler only determines whether or not a job can run on current resources when backfilling is used. If backfilling is turned off, then the scheduler won't determine this. It just decides it can't run now.

The scheduler logs this message for the top job only.

4.8.3.8.viii  Backfilling and Estimating Job Start Time

When the scheduler is backfilling around jobs, it estimates the start times and exec_vnode for the top jobs being backfilled around. See section 4.8.15, “Estimating Job Start Time”, on page 161.
4.8.3.8.ix Using Strict Ordering and Backfilling with Only One of Primetime or Non-primetime

When PBS is using strict ordering and backfilling, the scheduler saves a spot for each high-priority job around which it is backfilling. If you configure PBS to use strict ordering and backfilling for only one of primetime or non-primetime, and you have large jobs that must wait a long time before enough resources are available, the saved spots can be lost in the transition.

4.8.4 Examining Jobs Queue by Queue

When the scheduler examines waiting jobs, it can either consider all of the jobs in the complex as a whole, or it can consider jobs queue by queue. When considering jobs queue by queue, the scheduler runs all the jobs it can from the first queue before examining the jobs in the next queue, and so on. This behavior is controlled by the `by_queue` scheduler parameter.

When the `by_queue` scheduler parameter is set to `True`, jobs in the first queue are evaluated as a group, then jobs in the next queue are evaluated. In this case, PBS runs all the jobs it can from each queue before moving to the next queue, with the following exception: if there are jobs in the `Reservation`, `Express`, `Starving`, or `Suspended` job execution classes, those are considered before any queue. These classes are described in section 4.8.16, “Calculating Job Execution Priority”, on page 166.

The `by_queue` parameter applies to all of the queues in the complex. This means that either all jobs are scheduled as if they are in one large queue, or jobs are scheduled queue by queue.

If you want jobs to be examined queue by queue, in order of queue priority, you must also sort the queues and specify a different priority for each queue. To sort queues, set the `sort_queues` scheduler parameter to `True`. The `sort_queues` parameter is deprecated as of 12.2. To set queue priority, set each queue’s `priority` attribute to the desired value. A queue with a higher value is examined before a queue with a lower value. If you do not assign priorities to queues, their ordering is undefined. See “sort_queues” on page 295 of the PBS Professional Reference Guide, and “Queue Attributes” on page 354 of the PBS Professional Reference Guide.

The `by_queue` parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

See “by_queue” on page 283 of the PBS Professional Reference Guide.
4.8.4.1 Configuring PBS to Consider Jobs Queue by Queue

- Set the by_queue scheduler parameter to True
- Set the sort_queues scheduler parameter to True
- Assign a priority to each queue
- Choose whether you want queue by queue during primetime, non-primetime, or both. If you want separate behavior for primetime and non-primetime, list by_queue twice. For example:
  by_queue True prime
  by_queue False non_prime

The sort_queues parameter is deprecated as of 12.2.

4.8.4.2 Parameters and Attributes Affecting Queue by Queue

- The by_queue scheduler parameter; see “by_queue” on page 283 of the PBS Professional Reference Guide.
- The sort_queues scheduler parameter; see “sort_queues” on page 295 of the PBS Professional Reference Guide.
- The priority queue attribute; see “Queue Attributes” on page 354 of the PBS Professional Reference Guide.

4.8.4.3 Caveats and Advice for Queue by Queue

- The by_queue scheduler parameter is overridden by the round_robin scheduler parameter when round_robin is set to True.
- When by_queue is True, queues cannot be designated as fairshare entities, and fairshare will work queue by queue instead of on all jobs at once.
- When by_queue is True, job execution priority may be affected. See section 4.8.16, “Calculating Job Execution Priority”, on page 166.
- The by_queue parameter is not required when using express queues.
- You can have FIFO scheduling for all your jobs across the complex, if you are using a single execution queue or have by_queue set to False. However, you can have FIFO scheduling for the jobs within each queue if you set by_queue to True and specify a different priority for each queue. See section 4.8.19, “FIFO Scheduling”, on page 180.
4.8.5 Checkpointing

You can use checkpointing as a scheduling tool, by including it as a preemption method, an aid in recovery, a way to capture progress from a shrink-to-fit job, and when using the **qhold** command.

For a complete description of how to use and configure checkpointing, see section 9.3, “Checkpoint and Restart”, on page 711.

4.8.5.1 Checkpointing as a Preemption Method

When a job is preempted via checkpointing, MoM runs the **checkpoint_abort** script, and PBS kills and requeues the job. When the scheduler elects to run the job again, the MoM runs the restart script to restart the job from where it was checkpointed. See section 4.8.33, “Using Preemption”, on page 228.

4.8.5.2 Checkpointing as a Way to Capture Progress and Help Recover Work

When you use checkpointing to capture a job’s progress before the job is terminated, for example when a shrink-to-fit job’s wall time is exceeded, MoM runs the snapshot checkpoint script, and the job continues to run. See section 9.3, “Checkpoint and Restart”, on page 711.

4.8.5.3 Checkpointing When Using the **qhold** Command

When the **qhold** command is used to hold a checkpointable job, MoM runs the **checkpoint_abort** script, and PBS kills, requeues, and holds the job. A job with a hold on it must have the hold released via the **qrls** command in order to be eligible to run. For a discussion of the use of checkpointing for the **qhold** command, see section 9.3.7.6, “Holding a Job”, on page 731. For a description of the **qhold** command, see “**qhold**” on page 148 of the PBS Professional Reference Guide.

4.8.6 Organizing Job Chunks

You can specify how job chunks should be organized onto hosts or vnodes. Jobs can request their placement arrangement, and you can set defaults at queues and at the server to be inherited by jobs that do not request a placement. You can tell PBS to do the following:

- Put all chunks from a job onto a single host using the **place=pack** statement.
- Put each chunk on a separate host using the **place=scatter** statement. The number
of chunks must be fewer than or equal to the number of hosts.

- Put each chunk on a separate vnode using the `place=vscatter` statement. The number of chunks must be fewer than or equal to the number of vnodes.
- Put each chunk anywhere using the `place=free` statement.

To specify a placement default, set `resources_default.place=<arrangement>`, where `arrangement` is `pack`, `scatter`, `vscatter`, or `free`. For example, to have the default at QueueA be `pack`:

```
Qmgr: set queue QueueA resources_default.place=pack
```

You can specify that job chunks must be grouped in a certain way. For example, to require that chunks all end up on a shared router, use this:

```
place=group=router
```

For more about jobs requesting placement, see “Requesting Resources and Placing Jobs” on page 214 of the PBS Professional Reference Guide.

### 4.8.6.1 Caveats for Organizing Job Chunks

A placement specification for arrangement, sharing, and grouping is treated as one package by PBS. This means that if a job requests only one, any defaults set for the others are not inherited. For example, if you set a default of `place=pack:excl:group=router`, and a job requests only `place=pack`, the job does not inherit `excl` or `group=router`.

### 4.8.7 cron Jobs, or the Windows Task Scheduler

You can use `cron` jobs or the Windows Task Scheduler to make time-dependent modifications to settings, where you are scheduling according to time slots. For example, you can change settings for primetime and non-primetime configurations, making the following changes:

- Set nodes `offline` or not `offline`
- Change the number of `ncpus` on workstations
- Change the priority of queues, for example to change preemption behavior
- Start or stop queues
- Set primetime & non-primetime options

### 4.8.7.1 Caveats for cron Jobs and the Windows Task Scheduler

Make sure that your `cron` jobs or Windows Task Scheduler behave correctly when PBS is not running.
Be careful when changing available resources, such as when offline vnodes. You might prevent jobs from running that would otherwise run. For details, see section 4.6.2, “Jobs that Cannot Run on Current Resources”, on page 115.

4.8.8 Using Custom and Default Resources

The information in this section relies on understanding how jobs are allocated resources via inheriting defaults or via hooks. Before reading this section, please read section 11.2, “Allocating Resources to Jobs”, on page 821.

For complete details of how to configure and use custom resources, please see section 5.14, “Custom Resources”, on page 319.

You can use custom and default resources for several purposes:

- Routing jobs to the desired vnodes; see section 4.8.8.2, “Using Custom Resources to Route Jobs”, on page 134
- Assigning execution priority to jobs; see section 4.8.8.3, “Using Custom Resources to Assign Job Execution Priority”, on page 134
- Tracking and controlling the allocation of resources; see section 4.8.8.4, “Using Custom Resources to Track and Control Resource Allocation”, on page 135
- Representing elements such as GPUs, FPGAs, and switches; see section 4.8.8.5, “Using Custom Resources to Represent GPUs, FPGAs, Switches, Etc.”, on page 135
- Allowing users to request platform-specific resources, for example Cray-specific resources; see section 4.8.8.6, “Using Custom Resources to Allow Platform-specific Resource Requests”, on page 135
- Allowing users to submit jobs that run on a Cray as they would if using the aprun command; see section 4.8.8.7, “Using Custom Resources to Allow Platform-specific Behavior”, on page 135
- Shrinking job walltime so that they can run in time slots that are less than the expected maximum. See section 4.8.41, “Using Shrink-to-fit Jobs”, on page 263.

4.8.8.1 Techniques for Allocating Custom Resources to Jobs

In addition to using custom resources to represent physical elements such as GPUs, you can use custom resources as tags that you attach to jobs in order to help schedule the jobs. You can make these custom resources into tools that can be used only for managing jobs, by making them unalterable and unrequestable, and if desired, invisible to users.

For how to assign custom and default resources to jobs, see section 11.2, “Allocating Resources to Jobs”, on page 821.
4.8.8.2 Using Custom Resources to Route Jobs

You can use several techniques to route jobs to the desired queues and/or vnodes. Depending on your site’s configuration, you may find it helpful to use custom resources with one or more of these techniques.

• You can force users to submit jobs to the desired queues by setting resource limits at queues. You can use custom resources to represent arbitrary elements, for example, projects. In this case you could limit which project uses each queue. You can set a default value for the project at the server, or create a hook that assigns a value for the project.

For how queue resource limits are applied to jobs, see section 2.2.6.4.i, “How Queue and Server Limits Are Applied, Except Running Time”, on page 26.

• Use default resources or a hook to assign custom resources to jobs when the jobs are submitted. Send the jobs to routing queues, then route them, using the resources, to other queues inside or outside the PBS complex. Again, custom resources can represent arbitrary elements.

For how routing queues work, see section 2.2.6, “Routing Queues”, on page 24.

• Use peer scheduling to send jobs between PBS complexes. You can set resource limits on the furnishing queue in order to limit the kinds of jobs that are peer scheduled. You can assign custom resources to jobs to represent arbitrary elements, for example peer queueing only those jobs from a specific project. You can assign the custom resource by having the job inherit it or via a hook.

For how to set up peer scheduling, see section 4.8.31, “Peer Scheduling”, on page 205.

• You can route jobs from specific queues to the desired vnodes, by associating the vnodes with the queues. See section 4.8.2, “Associating Vnodes with Queues”, on page 121.

• You can create placement sets so that jobs are placed according to resource values. Placement sets are created where vnodes share a value for a resource; you can use custom resources to create the placement sets you want. See section 4.8.32, “Placement Sets”, on page 211.

4.8.8.3 Using Custom Resources to Assign Job Execution Priority

You can use custom resources as coefficients in the job sorting formula. You can assign custom resources to jobs using the techniques listed in section 11.2, “Allocating Resources to Jobs”, on page 821. The value of each custom resource can be based on a project, an application, etc.
For example, you can create a custom resource called “ProjPrio”, and the jobs that request the “Bio” project can be given a value of 5 for ProjPrio, and the jobs that request the “Gravel” project can be given a value of 2 for ProjPrio. You can assign this value in a hook or by routing the jobs through special queues from which the jobs inherit the value for ProjPrio.

For information on using the job sorting formula, see section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.

### 4.8.8.4 Using Custom Resources to Track and Control Resource Allocation

You can use resources to track and control usage of things like hardware and licenses. For example, you might want to limit the number of jobs using floating licenses or a particular vnode. See section 5.10, “Using Resources to Track and Control Allocation”, on page 314.

### 4.8.8.5 Using Custom Resources to Represent GPUs, FPGAs, Switches, Etc.

You can use custom resources to represent GPUs, FPGAs, high performance switches, etc. For examples, see section 5.14.8, “Using GPUs”, on page 360, section 5.14.9, “Using FPGAs”, on page 364, and section 10.2.7, “Allowing Users to Request HPS”, on page 776.

### 4.8.8.6 Using Custom Resources to Allow Platform-specific Resource Requests

PBS is integrated with Cray, and provides special custom resources to represent Cray resources. You can create other custom resources to represent other platform-specific elements. For an example, see section 10.3.7.13, “Allowing Users to Request Login Node Groups”, on page 795.

### 4.8.8.7 Using Custom Resources to Allow Platform-specific Behavior

You can create custom resources that allow Cray users to run jobs that behave the same way they would if the user had used the `aprun` command. For examples, see section 10.3.7.11, “Allowing Users To Reserve N NUMA Nodes Per Compute Node”, on page 791 and section 10.3.7.12, “Allowing Users To Reserve Specific NUMA Nodes”, on page 793.
4.8.9 Using Idle Workstation Cycle Harvesting

You can configure workstations at your site so that PBS can run jobs on them when their “owners” are away and they are idle. This is called *idle workstation cycle harvesting*. This can give your site additional resources to run jobs during nights and weekends.

You can configure PBS to use the following methods to decide when a workstation is not being used by its owner:

- Keyboard/mouse activity
- X-Window monitoring
- Load average (not recommended)

On some systems cycle harvesting is simple to implement, because the console, keyboard, and mouse device access times are periodically updated by the operating system. The PBS MoM process can track this information, and mark the vnode *busy* if any of the input devices is in use. On other systems, however, this data is not available: on some machines, PBS can monitor the X-Window system in order to obtain interactive idle time, and on others, PBS itself monitors keyboard and mouse activity.

Jobs on workstations that become *busy* are not migrated; they remain on the workstation until they complete execution, are rerun, or are deleted.

4.8.9.1 Platforms Supporting Cycle Harvesting

Due to different operating system support for tracking mouse and keyboard activity, the availability and method of support for cycle harvesting varies based on the computer platform in question. The following table lists the method and support for each platform.

<table>
<thead>
<tr>
<th>System</th>
<th>Status</th>
<th>Method</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>supported</td>
<td><code>pbs_idled</code></td>
<td>“Cycle Harvesting by Monitoring X-Windows” on page 146.</td>
</tr>
<tr>
<td>HP-UX 11</td>
<td>supported</td>
<td>keyboard/mouse</td>
<td>section 4.8.9.3, “Cycle Harvesting Based on Keyboard/Mouse Activity”, on page 138</td>
</tr>
<tr>
<td>Linux</td>
<td>supported</td>
<td>keyboard/mouse</td>
<td>section 4.8.9.3, “Cycle Harvesting Based on Keyboard/Mouse Activity”, on page 138</td>
</tr>
</tbody>
</table>
4.8.9.2 The $kbd_idle MoM Configuration Parameter

Cycle harvesting based on keyboard/mouse activity and X-Windows monitoring is controlled by the $kbd_idle MoM configuration parameter in PBS_HOME/mom_priv/config on the workstation in question. This parameter has the following format:

$kbd_idle <idle_wait> <min_use> <poll_interval>

Declares that the vnode will be used for batch jobs during periods when the keyboard and mouse are not in use.

idle_wait
Time, in seconds, that the workstation keyboard and mouse must be idle before being considered available for batch jobs.
Must be set to value greater than 0 for cycle harvesting to be enabled.
Format: Integer
No default

min_use
Time, in seconds, during which the workstation keyboard or mouse must continue to be in use before the workstation is determined to be unavailable for batch jobs.
Format: Integer
Default: 10

poll_interval
Interval, in seconds, at which MoM checks for keyboard and mouse activity.
Format: Integer
Default: 1

---

### Table 4-5: Cycle Harvesting Support Methods

<table>
<thead>
<tr>
<th>System</th>
<th>Status</th>
<th>Method</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris</td>
<td>supported</td>
<td>keyboard/mouse</td>
<td>section 4.8.9.3, “Cycle Harvesting Based on Keyboard/Mouse Activity”, on page 138</td>
</tr>
<tr>
<td>Windows</td>
<td>supported</td>
<td>keyboard/mouse</td>
<td>section 4.8.9.4, “Cycle Harvesting on Windows”, on page 139</td>
</tr>
</tbody>
</table>
4.8.9.3 Cycle Harvesting Based on Keyboard/Mouse Activity

PBS can monitor a workstation for keyboard and mouse activity, and run batch jobs on the workstation when the keyboard and mouse are not being used. PBS sets the state of the vnode to either free or busy, depending on whether or not there is keyboard or mouse activity, and runs jobs only when the state of the vnode is free. PBS sets the state of the vnode to free when the vnode’s mouse and keyboard have shown no activity for the specified amount of time. If PBS determines that the vnode is being used, it sets the state of the vnode to busy and suspends any running jobs, setting their state to U (user busy).

This method is used for Linux, Solaris, and HP operating systems.

4.8.9.3.i Configuring Cycle Harvesting Using Keyboard/Mouse Activity

To configure cycle harvesting using keyboard and mouse activity, do the following:

1. Set the $kbd_idle MoM configuration parameter by editing the $kbd_idle parameter in PBS_HOME/mom_priv/config on the workstation.

2. HUP the MoM on the workstation:

   `kill -HUP <pbs_mom PID>`

4.8.9.3.ii Example of Cycle Harvesting Using Keyboard/Mouse Activity

The following is an example setting for the parameter:

   `$kbd_idle 1800 10 5`

This setting for the parameter in MoM’s config file specifies the following:

- PBS marks the workstation as free if the keyboard and mouse are idle for 30 minutes (1800 seconds)
- PBS marks the workstation as busy if the keyboard or mouse is used for 10 consecutive seconds
- The states of the keyboard and mouse are to be checked for activity every 5 seconds
Here, we walk through how this example would play out, to show the roles of the arguments to the $kdb_idle parameter:

Let’s start with a workstation that has been in use for some time by its owner. The workstation is in state busy.

Now the owner goes to lunch. After 1800 seconds (30 minutes), PBS changes the workstation’s state to free and starts a job on the workstation.

Some time later, someone walks by and moves the mouse or enters a command. Within the next 5 seconds (idle poll period), pbs_mom notes the activity. The job is suspended and placed in state U, and the workstation is marked busy.

If 10 seconds pass and there is no additional keyboard/mouse activity, the job is resumed and the workstation again is either free (if any CPUs are available) or job-busy (if all CPUs are in use.)

However, if keyboard/mouse activity continues during that 10 seconds, the workstation remains busy and the job remains suspended for at least the next 1800 seconds.

4.8.9.3.iii Caveats for Cycle Harvesting Using Keyboard/Mouse Activity

• There is no default for idle_wait; you must set it to a value greater than 0 in order to enable cycle harvesting using keyboard/mouse activity.

4.8.9.4 Cycle Harvesting on Windows

A process called pbs_idled monitors keyboard and mouse activity and keeps MoM informed of user activity. The user being monitored can be sitting at the machine, or using a remote desktop.

The pbs_idled process is managed in one of two ways. PBS can use a service called PBS_INTERACTIVE to monitor the user’s session. If the PBS_INTERACTIVE service is registered, MoM starts the service, and the service starts and stops pbs_idled. The PBS_INTERACTIVE service runs under a local system account. PBS uses the PBS_INTERACTIVE service only where site policy allows a local system account to be a service account. If this is not allowed (so the service is not registered), pbs_idled is started and stopped using the log on/log off script. Do not use both the PBS_INTERACTIVE service and a log on/log off script.

A pbs_idled process monitors the keyboard and mouse activity while a user is logged in. This process starts when the user logs on, and stops when the user logs off. Only a user with administrator privileges, or the user being monitored, can stop pbs_idled.

MoM uses two files to communicate with pbs_idled:

• MoM creates PBS_HOME/spool/idle_poll_time and writes the value of her $kbd_idle polling interval parameter to it. The pbs_idled process reads the value of
the polling interval from `idle_poll_time`.

- MoM creates `PBS_HOME/spool/idle_touch`. The `pbs_idled` process updates the time stamp of the `idle_touch` file when a user is active, and MoM reads the time stamp.

### 4.8.9.4.i Configuring Cycle Harvesting on Windows

To configure cycle harvesting, do the following:

1. Make sure that you are a user with administrator privileges.

2. Set the `$kbd_idle` MoM configuration parameter by editing the `$kbd_idle` parameter in `PBS_HOME/mom_priv/config` on the workstation.

3. Configure how `pbs_idled` starts:
   
   a. If your policy allows a local system account to be a service account, register the PBS_INTERACTIVE service:
      ```
pbs_interactive -R
      ```

   b. If your policy does not allow a local system account to be a service account, and you are in a domained environment:
      
      1. Configure the log on script as described in section 4.8.9.4.ii, “Configuring `pbs_idled` in Log On Script in Domain Environment”, on page 141.
      
      2. Configure the log off script as described in section 4.8.9.4.iii, “Configuring `pbs_idled` in Log Off Script in Domain Environment”, on page 142.

   c. If your policy does not allow a local system account to be a service account, and you are in a standalone environment:
      
      1. Configure the log on script as described in section 4.8.9.4.iv, “Configuring `pbs_idled` in Log On Script in Standalone Environment”, on page 143.
      
      2. Configure the log off script as described in section 4.8.9.4.v, “Configuring `pbs_idled` in Log Off Script in Standalone Environment”, on page 144.

4. Restart the MoM.
4.8.9.4.ii Configuring \texttt{pbs\_idled} in Log On Script in Domain Environment

1. You must be a user with administrator privileges.
2. On the domain controller host, open \textit{Administrator Tools}.
3. In \textit{Administrator Tools}, open \textit{Active Directory Users and Computers}.
4. Right-click on the Organizational Unit where you want to apply the group policy for logging on and logging off.
5. Click on \textit{Properties}.
6. Go to the \textit{Group Policy} tab under the \textit{Properties} window.
7. Click on \textit{New}.
8. Type “LOG-IN-OUT-SCRIPT” as the name of the policy.
9. Select the Group Policy Object you have just created; click \textit{Edit}. The Group Policy Object editing window will open.
10. Open \textit{Window Settings} in \textit{User Configuration}.
11. Open \textit{Scripts (Logon/Logoff)}.
12. Open \textit{Logon}. A \textit{Logon Properties} window will open.
13. Open Notepad in another window. In Notepad, you create the command that starts the \texttt{pbs\_idled} process:
   ```
   pbs\_idled start
   ```
14. Save that document as “\texttt{pbs\_idled\_logon.bat}”.
15. In the \textit{Logon Properties} window, click on \textit{Show Files}. A logon script folder will open in a new window.
16. Copy \texttt{pbs\_idled\_logon.bat} into the logon script folder and close the logon script folder window.
17. In the \textit{Logon Properties} window, click on \textit{Add}, and then click on \textit{Browse}. Select \texttt{pbs\_idled\_logon.bat} and then click on \textit{Open}.
18. Click on \textit{OK}, then \textit{Apply}, then again \textit{OK}.
20. Close the \textit{Active Directory Users and Computers} window.
21. Close the \textit{Administrator Tools} window.
4.8.9.4.iii Configuring pbs\_idled in Log Off Script in Domain Environment

1. You must be a user with administrator privileges.
2. On the domain controller host, open Administrator Tools.
4. Right-click on the Organizational Unit where you want to apply the group policy for logging on and logging off.
5. Click on Properties.
6. Go to the Group Policy tab under the Properties window.
7. Click on New.
8. Type “LOG-IN-OUT-SCRIPT” as the name of the policy.
9. Select the Group Policy Object you have just created; click Edit. The Group Policy Object editing window will open.
10. Open Window Settings in User Configuration.
11. Open Scripts (Logon/Logoff).
13. Open Notepad in another window. In Notepad, you create the command that stops the pbs\_idled process:
    
    pbs\_idled stop

14. Save that document as “pbs\_idled_logoff.bat”.
15. In the Logoff Properties window, click on Show Files. A logoff script folder will open in a new window.
16. Copy pbs\_idled_logoff.bat into the logoff script folder and close the logoff script folder window.
17. In the Logoff Properties window, click on Add, and then click on Browse. Select pbs\_idled_logoff.bat and then click on Open.
18. Click on OK, then Apply, then again OK.
4.8.9.4.iv Configuring `pbs_idled` in Log On Script in Standalone Environment

1. You must be a user with administrator privileges.
2. As administrator, open a command prompt, and type the following command:
   ```
   gpedit.msc
   ```
3. Press Enter. A Local Group Policy editing window will open.
4. Open Window Settings in User Configuration.
5. Open Scripts (Logon/Logoff).
6. Open Logon. A Logon Properties window will open.
7. Open Notepad in another window. In Notepad, you create the command that starts the `pbs_idled` process:
   ```
   pbs_idled start
   ```
8. Save that document as “`pbs_idled_logon.bat`”.
9. In the Logon Properties window, click on Show Files. A logon script folder will open in a new window.
10. Copy `pbs_idled_logon.bat` into the logon script folder and close the logon script folder window.
11. In the Logon Properties window, click on Add, and then click on Browse. Select `pbs_idled_logon.bat` and then click on Open.
12. Click on OK, then Apply, then again OK.
13. Close the Local Group Policy editing window.
4.8.9.4.v Configuring `pbs_idled` in Log Off Script in Standalone Environment

1. You must be a user with administrator privileges.

2. As administrator, open a command prompt, and type the following command:

   `gpedit.msc`

3. Press Enter. A Local Group Policy editing window will open.

4. Open Window Settings in User Configuration.

5. Open Scripts (Logon/Logoff).

6. Open Logoff. A Logoff Properties window will open.

7. Open Notepad in another window. In Notepad, you create the command that stops the `pbs_idled` process:

   ```
   pbs_idled stop
   ```

8. Save that document as “`pbs_idled_logoff.bat`”.

9. In the Logoff Properties window, click on Show Files. A logoff script folder will open in a new window.

10. Copy `pbs_idled_logoff.bat` into the logoff script folder and close the logoff script folder window.

11. In the Logoff Properties window, click on Add, and then click on Browse. Select `pbs_idled_logoff.bat` and then click on Open.

12. Click on OK, then Apply, then again OK.

13. Close the Local Group Policy editing window.

4.8.9.4.vi The PBS_INTERACTIVE Service

The PBS_INTERACTIVE service starts the `pbs_idled` process, as the current user, in the current active user’s session. Each time a user logs on, the service starts a `pbs_idled` for that user, and when that user logs off, the service stops that user’s `pbs_idled` process.

The service runs under a local system account. If your policy allows a local system account to be a service account, you can use PBS_INTERACTIVE. Otherwise you must configure `pbs_idled` in log on/log off scripts.

If you have configured the `$kbd_idle` MoM parameter, and you have registered the service, MoM starts the service. The service cannot be started manually.
If you will use PBS_INTERACTIVE, you must register the service. The installer cannot register the service.

- To register the PBS_INTERACTIVE service:
  
  ```bash
  pbs_interactive -R
  ```

  Upon successful execution of this command, the following message is displayed:
  
  "Service PBS_INTERACTIVE installed successfully"

- To unregister the PBS_INTERACTIVE service:
  
  ```bash
  pbs_interactive -U
  ```

  Upon successful execution of this command, the following message is displayed:
  
  "Service PBS_INTERACTIVE uninstalled successfully"

- To see the version number for PBS_INTERACTIVE service:
  
  ```bash
  pbs_interactive --version
  ```

4.8.9.4.vii  Errors and Logging

If the $kbd_idle MoM parameter is configured, MoM attempts to use cycle harvesting. MoM looks for the PBS_INTERACTIVE service in the Service Control Manager. If she finds the service, she starts it.

1. If she cannot find the service, MoM logs the following message at event class 0x0002:

   "Can not find PBS_INTERACTIVE service, Continuing Cycle Harvesting with Logon/Logoff Script"

2. MoM looks for PBS_HOME/spool/idle_touch. If she finds it, she uses cycle harvesting.

3. If she cannot find the file, MoM disables cycle harvesting and logs the following message at event class 0x0002:

   "Cycle Harvesting Failed, Please contact Admin"
MoM logs the following messages at event class 0x0001.

- If MoM fails to open the Service Control Manager:
  "OpenSCManager failed for PBS_INTERACTIVE"
- If MoM fails to open the PBS_INTERACTIVE service:
  "OpenService failed for PBS_INTERACTIVE"
- If MoM fails to start the PBS_INTERACTIVE service:
  "Could not start PBS_INTERACTIVE service"
- If MoM fails to get status information about the PBS_INTERACTIVE service:
  "Can not get information about PBS_INTERACTIVE service"
- If MoM fails to send a stop control message to the PBS_INTERACTIVE service:
  "Could not stop PBS_INTERACTIVE service"
- If the PBS_INTERACTIVE service does not respond in a timely fashion:
  "PBS_INTERACTIVE service did not respond in timely fashion"
- If MoM fails to create idle_touch and idle_poll_time in PBS_HOME/spool directory:
  "Can not create file < full path of idle file >"
- If MoM fails to write the idle polling interval into PBS_HOME/spool/idle_poll_time:
  "Can not write idle_poll time into < full path of idle_poll_time file > file"

4.8.9.4.viii Caveats for Cycle Harvesting on Windows

- Under Windows, if the pbs_idled process is killed, cycle harvesting will not work.
- Under Windows, cycle harvesting may not work correctly on machines where more than one user is logged in, and users are not employing Switch User.
- Do not use both the PBS_INTERACTIVE service and a log on/log off script.

4.8.9.5 Cycle Harvesting by Monitoring X-Windows

On UNIX/Linux machines where the OS does not periodically update console, keyboard, and mouse device access times, PBS can monitor X-Window activity instead. PBS uses an X-Window monitoring process called pbs_idled. This process runs in the background and monitors X and reports to the pbs_mom whether or not the vnode is idle. pbs_idled is located in $PBS_EXEC/sbin.

This method is used for machines running AIX operating systems.
To configure PBS for cycle harvesting by monitoring X-Windows, perform the following steps:

1. Create a directory for `pbs_idled`. This directory must have the same permissions as `/tmp` (i.e. mode 1777). This will allow the `pbs_idled` program to create and update files as the user, which is necessary because the program runs as the user. For example:
   
   ```bash
   mkdir PBS_HOME/spool/idledir
   chmod 1777 PBS_HOME/spool/idledir
   ```

2. Turn on keyboard idle detection in the MoM config file:
   
   ```bash
   $kbd_idle <idle wait value>
   ```

3. Include `pbs_idled` as part of the X-Windows startup sequence.

   The best and most secure method of starting `pbs_idled` is via the system-wide `Xsession` file. This is the script which is run by `xdm` (the X login program) and sets up each user's X-Windows environment.

   You **must** place the startup line for `pbs_idled` before that of the window manager.

   You **must** make sure that `pbs_idled` runs in the background.

   On systems that use `Xsession` to start desktop sessions, insert a line invoking `pbs_idled` near the top of the file.

   For example, insert the following line in a Linux `Xsession` file:
   
   ```bash
   /usr/pbs/sbin/pbs_idled &
   ```

   If access to the system-wide `Xsession` file is not available, you can add `pbs_idled` to every user's personal `.xsession` or `.xinitrc` file, depending on the local OS requirements for starting X-windows programs upon login.

### 4.8.9.6 Cycle Harvesting Based on Load Average

Cycle harvesting based on load average means that PBS monitors each workstation’s load average, runs jobs where workstations have loads below a specified level, and suspends any batch jobs on workstations whose load has risen above the limit you set. When a workstation’s owner uses the machine, the workstation’s load rises.

When you configure cycle harvesting based on load average, you are performing the same configuration as for load balancing using load average. For a complete description of load balancing, see section 4.8.27, “Using Load Balancing”, on page 193.
4.8.9.6.i Attributes and Parameters Affecting Cycle Harvesting Based on Load Average

load_balancing
Scheduler parameter. When set to True, the scheduler places jobs only where the load average is below the specified limit.
- Format: Boolean
- Default: False

ideal_load <load>
MoM parameter. Defines the load below which the vnode is not considered to be busy. Used with the $max_load directive.
- Example:
  $ideal_load 1.8
- Format: Float
- No default

max_load <load> [suspend]
MoM parameter. Defines the load above which the vnode is considered to be busy. Used with the $ideal_load directive. No new jobs are started on a busy vnode.
The optional suspend directive tells PBS to suspend jobs running on the node if the load average exceeds the $max_load number, regardless of the source of the load (PBS and/or logged-in users). Without this directive, PBS will not suspend jobs due to load.
We recommend setting this to a slightly higher value than your target load (which is typically the number of CPUs), for example .25 + ncpus.
- Example:
  $max_load 3.25
- Format: Float
- Default: number of CPUs

resv_enable
Vnode attribute. Controls whether the vnode can be used for advance and standing reservations. When set to True, this vnode can be used for reservations.
- Format: Boolean
- Default: True
4.8.9.6.ii How Cycle Harvesting Based on Load Average Works

Cycle harvesting based on load average means that PBS monitors the load average on each machine. When the load on a workstation is below what is specified in the $ideal_load MoM parameter, PBS sets the state of the workstation to free. The scheduler will run jobs on vnodes whose state is free. When the load on a workstation exceeds the setting for $max_load, PBS sets the state of the workstation to busy, and suspends jobs running on the workstation. PBS does not start jobs on a vnode whose state is busy. When the load drops below the setting for $ideal_load, PBS sets the state to free, and resumes the jobs that were running on the workstation.

PBS thinks that a 1-CPU job raises a vnode’s load by 1. On machines being used for cycle harvesting, values for $max_load and $ideal_load are set to reasonable limits. On other machines, these are set to values that will never be exceeded, so that load is effectively ignored.

On machines where these parameters are unset, the vnode’s state is not set according to its load, so jobs are not suspended because a vnode is busy. However, if $max_load and $ideal_load are unset, they are treated as if they have the same value as resources_available.ncpus, and because there is usually a small background load, PBS will lose the use of a CPU’s worth of load. The scheduler won’t place a job where the anticipated load would exceed $max_load, so if a machine has a load of 1.25, is running a 1-CPU job, and has 2 CPUs, PBS won’t place another 1-CPU job there.

4.8.9.6.iii Configuring Cycle Harvesting Based on Load Average

To set up cycle harvesting for idle workstations based on load average, perform the following steps:

1. If PBS is not already installed on the target execution workstations, do so now, selecting the execution-only install option. See the PBS Professional Installation & Upgrade Guide.

2. Edit the PBS_HOME/mom_priv/config configuration file on each target execution workstation, adding the $max_load and $ideal_load configuration parameters. Make
sure they have values that will not interfere with proper operation. See section 4.8.9.6.v, “Caveats for Cycle Harvesting Based on Load Average”, on page 150.

$max_load <load limit that allows jobs to run>
$ideal_load <load at which to start jobs>

3. Edit the PBS_HOME/mom_priv/config configuration file on each machine where you are not using cycle harvesting, adding the $max_load and $ideal_load configuration parameters. Make sure they have values that will never be exceeded.

$max_load <load limit that will never be exceeded>
$ideal_load <load limit that will never be exceeded>

4. HUP the MoM:

   kill -HUP <pbs_mom PID>

5. Edit the PBS_HOME/sched_priv/sched_config configuration file to direct the Scheduler to perform scheduling based on load_balancing.

   load_balancing: True   ALL

6. If you wish to oversubscribe the vnode’s CPU(s), set its resources_available.ncpus to a higher number. Do this only on single-vnode machines. You must be cautious about matching ncpus and $max_load. See "Caveats for Cycle Harvesting Based on Load Average" on page 150 in the PBS Professional Administrator’s Guide.

7. HUP the scheduler:

   kill -HUP <pbs_sched PID>

8. Set the vnode’s resv_enable attribute to False, to prevent the workstation from being used for reservations.

   Qmgr: set node <vnode name> resv_enable = False

9. Set the vnode’s no_multinode_jobs attribute to True, to prevent the workstation from stalling multichunk jobs.

   Qmgr: set node <vnode name> no_multinode_jobs = True

4.8.9.6.iv Viewing Load Average Information

You can see the state of a vnode using the pbsnodes -a command.

4.8.9.6.v Caveats for Cycle Harvesting Based on Load Average

• Be careful with the settings for $ideal_load and $max_load. You want to make sure that when the workstation owner is using the machine, the load on the machine triggers MoM
to report being busy, and that PBS does not start any new jobs while the user is working.

- For information about keeping your site running smoothly using $\text{max\_load}$ and $\text{ideal\_load}$, see section 9.4.5, “Managing Load Levels on Vnodes”, on page 738.
- If you set ncpus higher than the number of actual CPUs, and set $\text{max\_load}$ higher to match, keep in mind that the workstation user could end up with an annoyingly slow workstation. This can happen when PBS runs jobs on the machine, but the combined load from the jobs and the user is insufficient for MoM to report being busy.

4.8.9.7 Cycle Harvesting and File Transfers

The cycle harvesting feature interacts with file transfers in one of two different ways, depending on the method of file transfer:

- If the user’s job includes file transfer commands (such as rcp or scp) within the job script, and such a command is running when PBS decides to suspend the job on the vnode, then the file transfer is suspended as well.
- If the job has PBS file staging parameters (i.e. stagein=, stageout=file1...), and the load goes above $\text{max\_load}$, the file transfer is not suspended. This is because the file staging is not part of the job script execution, and is not subject to suspension. See "Detailed Description of Job’s Lifecycle", on page 58 of the PBS Professional User’s Guide.

4.8.9.8 Parallel Jobs With Cycle Harvesting

Cycle harvesting is not recommended for hosts that will run multi-host jobs. However, you may find that your site benefits from using cycle harvesting on these machines. We provide advice on how to prevent cycle harvesting on these machines, and advice on how to accomplish it.

4.8.9.8.i General Advice: Parallel Jobs Not Recommended

Cycle harvesting is somewhat incompatible with multi-host jobs. If one of the hosts being used for a parallel job running on several hosts is being used for cycle harvesting, and the user types at the keyboard, job execution will be delayed for the entire job because the tasks running on that host will be suspended.

To prevent a machine which is being used for cycle harvesting from being assigned a multi-host job, set the vnode’s no_multinode_jobs attribute to True. This attribute prevents a host from being used by jobs that span multiple hosts.
4.8.9.8.ii How to Use Cycle Harvesting with Multi-host Jobs

When a single-host job is running on a workstation configured for cycle harvesting, and that host becomes busy, the job is suspended. However, suspending a multi-host parallel job may have undesirable side effects because of inter-process communications. For a job which uses multiple hosts when one or more of the hosts becomes busy, the default action is to leave the job running.

However, you can specify that the job should be requeued and subsequently re-scheduled to run elsewhere when any of the hosts on which the job is running becomes busy. To enable this action, add the following parameter to MoM’s configuration file:

```
$action multinodebusy 0 requeue
```

where `multinodebusy` is the action to modify; “0” (zero) is the action timeout value (it is ignored for this action); and `requeue` is the new action to perform. The only action that can be performed is requeuing.

Multi-host jobs which are not rerunnable (i.e. those submitted with the `qsub -rn` option) will be killed if the `requeue` argument is configured for the `multinodebusy` action and a vnode becomes busy.

4.8.9.9 Cycle Harvesting Caveats and Restrictions

4.8.9.9.i Cycle Harvesting and Multi-host Jobs

Cycle harvesting is not recommended for hosts that will run multi-host jobs. See section 4.8.9.8.i, “General Advice: Parallel Jobs Not Recommended”, on page 151.

4.8.9.9.ii Cycle Harvesting and Reservations

Cycle harvesting is incompatible with jobs in reservations. Reservations should not be made on a machine used for cycle harvesting, because the user may appear during the reservation period and use the machine’s keyboard. This will suspend the jobs in the reservation, defeating the purpose of making a reservation.

To prevent a vnode which is being used for cycle harvesting from being used for reservations, set the vnode’s `resv_enable` attribute to `False`. This attribute controls whether the vnode can be used for reservations.

4.8.9.9.iii File Transfers with Cycle Harvesting

4.8.9.9.iv  Cycle Harvesting on Windows

• Under Windows, if the `pbs_idled` process is killed, cycle harvesting will not work.
• Under Windows, cycle harvesting may not work correctly on machines where more than one user is logged in.

4.8.10  Dedicated Time

PBS provides a feature called *dedicated time* which allows you to define times during which the only jobs that can run are the ones in dedicated queues. You can use dedicated time for things like upgrades.

You can define multiple dedicated times. Any job in a dedicated time queue must have a wall-time in order to run. Jobs without walltimes will never run. PBS won’t let a reservation conflict with dedicated time. Hooks should not access or modify the dedicated time file.

For information on configuring dedicated time queues, see section 2.2.5.2.i, “Dedicated Time Queues”, on page 23.

4.8.10.1  Dedicated Time File

You define dedicated time by adding one or more time slots in the file `PBS_HOME/sched_priv/dedicated_time`. A time slot is a start date and start time and an end date and end time. Format:

```
<start date> <start time> <end date> <end time>
```

expressed as

```
MM/DD/YYYY HH:MM MM/DD/YYYY HH:MM
```

Any line whose first non-whitespace character is a pound sign (“#”) is a comment.

Example:

```
#Dedicated time for maintenance
04/15/2007 12:00 04/15/2007 15:30
```

A sample dedicated time file (`PBS_HOME/sched_priv/dedicated_time`) is included in the installation.

The dedicated time file is read on startup and HUP.
4.8.10.2 Steps in Defining Dedicated Time

You define dedicated time by performing the following steps:

1. Edit the file `PBS_HOME/sched_priv/dedicated_time` and add one or more time slots.

2. HUP the scheduler:
   - UNIX/Linux:
     ```bash
     kill -HUP <pbs_sched PID>
     ```
   - Windows:
     ```bash
     net stop pbs_sched
     net start pbs_sched
     ```

4.8.10.3 Recommendations for Dedicated Time

If you need to set up dedicated time for something like system maintenance, you may want to avoid having the machines become idle for a significant period before dedicated time starts. You can allow jobs to shrink their walltimes to fit into those shorter-than-normal slots before dedicated time. See section 4.8.41, “Using Shrink-to-fit Jobs”, on page 263.

4.8.11 Dependencies

PBS allows job submitters to specify dependencies between jobs, for example specifying that job J2 can only run if job J1 finishes successfully. You can add dependencies to jobs via a hook, default arguments to `qsub`, or via the `qalter` command.

For a description of how job dependencies work, see "Using Job Dependencies", on page 158 of the PBS Professional User’s Guide.

For how to use hooks, see section, “Hooks”, on page 413.

For how to add default `qsub` arguments, see “Server Attributes” on page 316 of the PBS Professional Reference Guide.

For how to use the `qalter` command, see “qalter” on page 127 of the PBS Professional Reference Guide.

4.8.12 Dynamic Resources

You can use dynamic PBS resources to represent elements that are outside of the control of PBS, typically for licenses and scratch space. You can represent elements that are available to the entire PBS complex as server-level resources, or elements that are available at a specific
host or hosts as host-level resources. For an example of configuring a server-level dynamic resource, see section 5.14.4.1.i, “Example of Configuring Dynamic Server-level Resource”, on page 335. For an example of configuring a dynamic host-level resource, see section 5.14.5.1.i, “Example of Configuring Dynamic Host-level Resource”, on page 338.

For a complete description of how to create and use dynamic resources, see section 5.14, “Custom Resources”, on page 319.

4.8.13 Eligible Wait Time for Jobs

PBS provides a method for tracking how long a job that is eligible to run has been waiting to run. By “eligible to run”, we mean that the job could run if the required resources were available. The time that a job waits while it is not running can be classified as “eligible” or “ineligible”. Roughly speaking, a job accrues eligible wait time when it is blocked due to a resource shortage, and accrues ineligible wait time when it is blocked due to project, user, or group limits. A job can only accrue one kind of wait time at a time, and cannot accrue wait time while it is running.

4.8.13.1 Types of Time Accrued

eligible_time
Job attribute. The amount of wall clock wait time a job has accrued because the job is blocked waiting for resources, or any other reason not covered by ineligible_time. For a job currently accruing eligible_time, if we were to add enough of the right type of resources, the job would start immediately. Viewable via qstat -f by job owner, Manager and Operator. Settable by Operator or Manager.

ineligible_time
The amount of wall clock time a job has accrued because the job is blocked by limits on the job’s project, owner, or group, or because the job is blocked because of its state.

run_time
The amount of wall clock time a job has spent running.

exiting
The amount of wall clock time a job has spent exiting.

initial_time
The amount of wall clock wait time a job has accrued before the type of wait time has been determined.
4.8.13.2 How Eligible Wait Time Works

A job accrues ineligible_time while it is blocked by project, user, or group limits, such as:

- max_run
- max_run_soft
- max_run_res.<resource>
- max_run_res_soft.<resources>

A job also accrues ineligible_time while it is blocked due to a user hold or while it is waiting for its start time, such as when submitted via

```
qsub -a <run-after> ...
```

A job accrues eligible_time when it is blocked by a lack of resources, or by anything not qualifying as ineligible_time or run_time. A job’s eligible_time will only increase during the life of the job, so if the job is requeued, its eligible_time is preserved, not set to zero. The job’s eligible_time is not recalculated when a job is qmove or moved due to peer scheduling.

For information on project, user, and group limits, see section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367.

The kind of time a job is accruing is sampled periodically, with a granularity of seconds.

A job’s eligible_time attribute can be viewed via qstat -f.

4.8.13.3 Configuring Eligible Wait Time

To enable using eligible time as the job’s wait time, set the eligible_time_enable server attribute to True.

4.8.13.4 How Eligible Wait Time Is Used

- You can choose to use each job’s eligible wait time as the amount of time it is starving. See section 4.8.46, “Starving Jobs”, on page 279.
- When a job is requeued, for example being checkpointed and aborted or preempted, its accumulated queue waiting time depends on how that time is calculated:
  - If you are using eligible time, the accumulated waiting time is preserved
  - If you are not using eligible time, the accumulated waiting time is lost

4.8.13.5 Altering Eligible Time

You can set the value for a job’s `eligible_time` attribute using the `qalter` command, for example:

```
qalter -Weligible_time=<time> <job ID>
```

4.8.13.6 Attributes Affecting Eligible Time

`eligible_time_enable`
Server attribute. Enables accumulation of eligible time for jobs. Controls whether a job’s `eligible_time` attribute is used as its starving time. See section 4.8.46, “Starving Jobs”, on page 279.

On an upgrade from versions of PBS prior to 9.1 or on a fresh install, `eligible_time_enable` is set to `False` by default.

When `eligible_time_enable` is set to `False`, PBS does not track `eligible_time`. Whether `eligible_time` continues to accrue for a job or not is undefined. The output of `qstat -f` does not include `eligible_time` for any job. Accounting logs do not show `eligible_time` for any job submitted before or after turning `eligible_time_enable` off. Log messages do not include accrual messages for any job submitted before or after turning `eligible_time_enable` off. If the scheduling formula includes `eligible_time`, `eligible_time` evaluates to 0 for all jobs.

When `eligible_time_enable` is changed from `False` to `True`, jobs accrue `eligible_time` or `ineligible_time` or `run_time` as appropriate. A job’s `eligible_time` is used for starving calculation starting with the next scheduling cycle; changing the value of `eligible_time_enable` does not change the behavior of an active scheduling cycle.

`accrue_type`
Job attribute. Indicates what kind of time the job is accruing.

### Table 4-6: Job’s `accrue_type` Attribute

<table>
<thead>
<tr>
<th>Type</th>
<th>Numeric Representation</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB_INITIAL</td>
<td>0</td>
<td>initial_time</td>
</tr>
<tr>
<td>JOB_INELIGIBLE</td>
<td>1</td>
<td>ineligible_time</td>
</tr>
</tbody>
</table>
The job’s `accrue_type` attribute is visible via `qstat` only by Manager, and is set only by the server.

**eligible_time**

Job attribute. The amount of wall clock wait time a job has accrued because the job is blocked waiting for resources, or any other reason not covered by `ineligible_time`. For a job currently accruing `eligible_time`, if we were to add enough of the right type of resources, the job would start immediately. Viewable via `qstat -f` by job owner, Manager and Operator. Settable by Operator or Manager.

### 4.8.13.7 Logging

The server prints a log message every time a job changes its `accrue_type`, with both the new `accrue_type` and the old `accrue_type`. These are logged at the 0x0400 event class.

Server logs for this feature display the following information:

- Time accrued between samples
- The type of time in the previous sample, which is one of initial time, run time, eligible time or ineligible time
- The next type of time to be accrued, which is one of run time, eligible time or ineligible time
- The eligible time accrued by the job, if any, until the current sample

<table>
<thead>
<tr>
<th>Type</th>
<th>Numeric Representation</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB_ELIGIBLE</td>
<td>2</td>
<td>eligible_time</td>
</tr>
<tr>
<td>JOB_RUNNING</td>
<td>3</td>
<td>run_time</td>
</tr>
<tr>
<td>JOB_EXIT</td>
<td>4</td>
<td>exit_time</td>
</tr>
</tbody>
</table>

Table 4-6: Job’s `accrue_type` Attribute
Example:

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 0 secs of initial_time, new accrue_type=eligible_time, eligible_time=00:00:00

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 1821 secs of eligible_time, new accrue_type=ineligible_time, eligible_time=01:20:22

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 2003 secs of ineligible_time, new accrue_type=eligible_time, eligible_time=01:20:22

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 61 secs of eligible_time, new accrue_type=run_time, eligible_time=01:21:23

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 100 secs of run_time, new accrue_type=ineligible_time, eligible_time=01:21:23

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 33 secs of ineligible_time, new accrue_type=eligible_time, eligible_time=01:21:23

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 122 secs of eligible_time, new accrue_type=run_time, eligible_time=01:23:25

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 1210 secs of run_time, new accrue_type=exiting, eligible_time=01:23:25

The example shows the following changes in time accrual:

- initial to eligible
- eligible to ineligible
- ineligible to eligible
- eligible to running
- running to ineligible
- ineligible to eligible
- eligible to running
- running to exiting

The server also logs the change in accrual when the job’s eligible_time attribute is altered using qalter. For example, if the job’s previous eligible time was 123 seconds, and it has been altered to be 1 hour and 1 minute:

Accrue type is eligible_time, previous accrue type was eligible_time for 123 secs, due to qalter total eligible_time=01:01:00
4.8.13.8 Accounting

Each job’s eligible_time attribute is included in the “E” and “R” records in the PBS accounting logs. See “Record Types” on page 420 of the PBS Professional Reference Guide.

Example:

08/07/2007 19:34:06;E;182.Host1;user=user1 group=user1 jobname=STDIN
queue=workq ctime=1186494765 qtime=1186494765 etime=1186494765
start=1186494767 exec_host=Host1/0 exec_vnode=(Host1:ncpus=1)
Resource_List.ncpus=1 Resource_List.nodect=1 Resource_List.place=pack
Resource_List.select=1:ncpus=1 session=4656 end=1186495446
Exit_status=-12 resources_used.cpupercent=0
resources_used.cput=00:00:00 resources_used.mem=3072kb
resources_used.ncpus=1 resources_used.vmem=13356kb
resources_used.walltime=00:11:21 eligible_time=00:10:00

4.8.14 Sorting Jobs by Entity Shares (Was Strict Priority)

You can sort jobs according to how much of the fairshare tree is allocated to the entity that owns the job. The fairshare percentages in the fairshare tree describe each entity’s share. Using entity shares is sorting jobs on a key, using the fair_share_perc option to the job_sort_key scheduler parameter.

Using entity shares, the jobs from an entity with greater allocation in the fairshare tree run before the jobs with a smaller allocation.

4.8.14.1 Configuring Entity Shares

To configure entity shares, do the following:

- Define fairshare tree entity allocation in PBS_HOME/sched_priv/resource_group. See section 4.8.18, “Using Fairshare”, on page 170. You can use a simple fairshare tree, where every entity’s parent_group is root.
- Give each entity shares according to desired priority, with higher-priority entities getting larger allocations.
- Set the unknown_shares scheduler parameter to 1. This causes any entity not in
your list of approved entities to have a tiny allocation, and the lowest priority.

For example:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Alloc</th>
<th>User</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>usr1</td>
<td>60</td>
<td>root</td>
<td>5</td>
</tr>
<tr>
<td>usr2</td>
<td>61</td>
<td>root</td>
<td>15</td>
</tr>
<tr>
<td>usr3</td>
<td>62</td>
<td>root</td>
<td>15</td>
</tr>
<tr>
<td>usr4</td>
<td>63</td>
<td>root</td>
<td>10</td>
</tr>
<tr>
<td>usr5</td>
<td>64</td>
<td>root</td>
<td>25</td>
</tr>
<tr>
<td>usr6</td>
<td>65</td>
<td>root</td>
<td>30</td>
</tr>
</tbody>
</table>

- Set `fair_share_perc` as the option to `job_sort_key`, for example:

  `job_sort_key: “fair_share_perc HIGH all”`

### 4.8.14.2 Viewing Entity Shares

When you are root, you can use the `pbsfs` command to view the fairshare tree allocations.

### 4.8.15 Estimating Job Start Time

PBS can estimate when jobs will run, and which vnodes each job will use. PBS estimates job start times and vnodes for all jobs using an asynchronous process, not the PBS server, scheduler, or MoM daemons.

Jobs have an attribute called `estimated` for reporting estimated start time and estimated vnodes. This attribute reports the values of two read-only built-in resources, `start_time` and `exec_vnode`. Each job’s estimated start time is reported in `estimated.start_time`, and its estimated vnodes are reported in `estimated.exec_vnode`.

PBS automatically sets the value of each job’s `estimated.start_time` value to the estimated start time for each job.

#### 4.8.15.1 Configuring Start Time Estimation

PBS estimates values for `start_time` and `exec_vnode` for jobs in the following ways:

- When `est_start_time_freq` is set to a value greater than zero, PBS estimates values for all jobs at the specified interval
- When `est_start_time_freq` is set to zero, PBS estimates values for all jobs after each scheduling cycle
- When the scheduler is backfilling around top jobs, it estimates the start times and `exec_vnode` for those jobs being backfilled around
If you want PBS to estimate the start time for all jobs, either set the `est_start_time_freq` server attribute to the interval at which you want PBS to make the calculation, or set it to zero, and the calculation will be made every scheduling cycle.

You can choose to have estimated start times for just the jobs being backfilled around. You set the number of jobs to be backfilled around by setting the server’s `backfill_depth` attribute to the desired number. See section 4.8.3, “Using Backfilling”, on page 124.

You can also choose to have `start_time` and `exec_vnode` estimated both for jobs being backfilled around and for all jobs.

Example 4-1: To estimate start times for the top 5 jobs every scheduling cycle, and for all jobs every 3 hours:

```
qmgr -c 'set server backfill_depth=5'
qmgr -c 'set server est_start_time_freq = 3:00:00'
```

### 4.8.15.2 Controlling User Access to Start Times and Vnode List

#### 4.8.15.2.i Making Start Time or Vnodes Invisible

You can make job estimated start times and vnodes invisible to unprivileged users by adding resource permission flags to the `start_time` or `exec_vnode` resources. To do this, add the resource to the `resourcedef` file, and include the i flag, in the same way you would for a custom resource being made invisible. Restart the server; see section 5.14.3.1, “Restarting the Server”, on page 332.

Example of making `start_time` and `exec_vnode` invisible to users:

```
start_time type=long flag=i
exec_vnode type=string flag=i
```

You can always make the start time and vnodes visible again to unprivileged users by removing the entries (or flags) in the `resourcedef` file and restarting the server.

See section 5.14.2.6, “Resource Permission Flags”, on page 328.

#### 4.8.15.2.ii Allowing Users to See Only Their Own Job Start Times

If you want users to be able to see the start times for their own jobs, but not those of other users, set the server’s `query_other_jobs` attribute to `False`, and do not set the i or r permission flags. Setting the server’s `query_other_jobs` attribute to `False` prevents a user from seeing anything about other users’ jobs.
4.8.15.3 Attributes and Parameters Affecting Job Start Time Estimation

backfill
Scheduler parameter.
Toggle that controls whether PBS uses backfilling. If this is set to True, the scheduler attempts to schedule smaller jobs around higher-priority jobs when using strict_ordering, as long as running the smaller jobs won't change the start time of the jobs they were scheduled around. The scheduler chooses jobs in the standard order, so other high-priority jobs will be considered first in the set to fit around the highest-priority job.

If this parameter is True, the scheduler backfills around starving jobs when help_starving_jobs is True.
PBS calculates estimated.start_time and estimated.exec_vnode for these jobs at each scheduler iteration.
Can be used with strict_ordering and help_starving_jobs.
Format: Boolean
Default: True all

backfill_depth
Server attribute.
Modifies backfilling behavior. Only used when server’s backfill attribute is True. Sets the number of jobs that are to be backfilled around.
Recommendation: set this to less than 100.
Format: Integer
Default: 1

estimated
List of values associated with job’s estimated start time. Used to report jobs' start_time and exec_vnode information. Can be set in a hook or via qalter, but PBS will overwrite the values. Allowable values: start_time, exec_vnode
Format: estimated.start_time, estimated.exec_vnode
Default: Unset

Python attribute value type: Dictionary: estimated.<resource name>=<value> where <resource name> is any resource
Chapter 4  

Scheduling

**est_start_time_freq**
Server attribute.
Interval at which PBS calculates estimated start times and vnodes for all jobs.
Best value is workload-dependent. Recommendation: set this to two hours.
When set to 0, PBS estimates start times for all jobs.
Format: *Duration*
Default: *Unset*
See “Server Attributes” on page 316 of the PBS Professional Reference Guide.

**help_starving_jobs**
Scheduler parameter.
Setting this option enables starving job support. Once jobs have waited for the amount of time given by *max_starve* they are considered starving. If a job is considered starving, then no lower-priority jobs will run until the starving job can be run, unless backfilling is also specified. To use this option, the *max_starve* configuration parameter needs to be set as well. See also *backfill*, *max_starve*, and the server’s *eligible_time_enable* attribute.
Format: *Boolean*
Default: *True all*

**strict_ordering**
Specifies that jobs must be run in the order determined by whatever sorting parameters are being used. This means that a job cannot be skipped due to resources required not being available. If a job due to run next cannot run, no job will run, unless backfilling is used, in which case jobs can be backfilled around the job that is due to run next.

Example line in PBS_HOME/sched_priv/sched_config:
```
strict_ordering: True ALL
```
Format: Boolean
Default: *False all*

### 4.8.15.4 Viewing Estimated Start Times
You can view the estimated start times and vnodes of jobs using the *qstat* command. If you use the -T option to *qstat* when viewing job information, the *Est Start* field is displayed. Running jobs are shown above queued jobs.
See “*qstat*” on page 196 of the PBS Professional Reference Guide.
If the estimated start time or vnode information is invisible to unprivileged users, no estimated start time or vnode information is available via `qstat`.

Example output:

```
qstat -T

Job ID   Username  Queue  Jobname  SessID  NDS  TSK  Memory  Time  S  Start
-------  --------  ------  --------  ------  ---  ---  ------  ----- - ----- 
5.host1  user1    workq  foojob  12345   1   1   128mb  00:10 R  --
9.host1  user1    workq  foojob   --    1   1   128mb  00:10 Q  11:30
10.host1 user1    workq  foojob   --    1   1   128mb  00:10 Q  Tu 15
7.host1  user1    workq  foojob   --    1   1   128mb  00:10 Q  Jul
8.host1  user1    workq  foojob   --    1   1   128mb  00:10 Q  2010
11.host1 user1    workq  foojob   --    1   1   128mb  00:10 Q  >5yrs
13.host1 user1    workq  foojob   --    1   1   128mb  00:10 Q  --
```

### 4.8.15.5 Selecting Jobs By Estimated Start Time

You can use the `qselect` command to select jobs according to their start times by using the `-t` suboption to the `-t` option. This selects jobs according to the value of the `estimated.start_time` attribute. See "`qselect` on page 185 of the PBS Professional Reference Guide."

### 4.8.15.6 Logging

Whenever the scheduler estimates the start time of a job, it logs the start time. The scheduler does not log the estimated `exec_vnode` of a job.

### 4.8.15.7 Caveats and Advice

- The `estimated.start_time` of a job array is the time calculated for the first queued subjob only.

- Cached estimated start times are only as fresh as the last time PBS calculated them. This should be taken into account when setting the values of `est_start_time_freq` and `backfill_depth`.

- The frequency of calculating start times is a trade-off between having more current start time information and using fewer computing cycles for non-job work. The background
Chapter 4  Scheduling

The task of calculating start times can be computationally intensive. This should be taken into account when setting the value of est_start_time_freq. Depending on the size of your site, it is probably a good idea not to set it to less than 10 minutes.

- The best value for est_start_time_freq is workload dependent, but we recommend setting it to two hours as a starting point.
- If your site has short scheduling cycles of a few minutes, and can use backfilling (and at least one of strict ordering or starving jobs), you can have the start times for all jobs calculated at each scheduling cycle. To do this, set backfill_depth to a value greater than the number of jobs the site will ever have, and do not set est_start_time_freq.
- We recommend setting backfill_depth to a value that is less than 100.

4.8.16  Calculating Job Execution Priority

When the scheduler examines jobs, either at the whole complex or within a queue, it gives each job an execution priority, and then uses this job execution priority to select which job(s) to run. Job execution priority is mostly independent of job preemption priority. We discuss only job execution priority in this section.

Some of the scheduler’s policy for determining job execution priority is built into PBS, but you can specify how execution priority is determined for most of the policy.

First, the scheduler divides queued jobs into classes. Then it sorts the jobs within each class.

4.8.16.1  Dividing Jobs Into Classes

PBS has special classes that supersede queue order, meaning that whether or not queues are being examined separately, the jobs in each of those classes are treated as if they are in a single queue. They are not ordered according to which queue they reside in. For example, all starving jobs are treated as if they are in a single queue. PBS has one class for all non-special jobs. This class typically contains most PBS jobs. Queue order is imposed on this class, meaning that if queues are ordered and sorted, jobs are examined queue by queue.
Job execution classes have a built-in order of precedence. All jobs in the highest class are considered before any jobs in the next class, and so on. Classes are listed in the following table, highest first:

### Table 4-7: Job Execution Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Sort Applied Within Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservation</td>
<td>Jobs submitted to an advance or standing reservation</td>
<td>Formula or job sort key</td>
</tr>
<tr>
<td>Express</td>
<td>All jobs with preemption priority higher than normal jobs. Preemption priority is defined in scheduler’s preempt_prio parameter. Jobs are sorted into this class only when preemption is enabled. See section 4.8.33, “Using Preemption”, on page 228.</td>
<td>First by preemption priority, then by formula or job sort key</td>
</tr>
<tr>
<td>Starving</td>
<td>Starving jobs. Jobs are sorted into this class only when starving is enabled. See section 4.8.46, “Starving Jobs”, on page 279</td>
<td>Amount of time counted toward starving</td>
</tr>
<tr>
<td>Suspended</td>
<td>Jobs that have been preempted via suspension, and jobs that were both suspended and resumed by hand.</td>
<td>Formula or job sort key</td>
</tr>
<tr>
<td>Normal</td>
<td>Jobs that do not belong in any of the special classes</td>
<td>Queue order, if it exists, then formula, fairshare, or job sort key</td>
</tr>
</tbody>
</table>

#### 4.8.16.2 Selecting Job Execution Class

The scheduler places each job in the highest-priority class into which the job can fit. So, for example, if a job is both in a reservation and is starving, the job is placed in the Reservation class.

#### 4.8.16.3 Sorting Jobs Within Classes

Jobs within each class are sorted according to rules specific to each class. The sorting applied to each class is listed in Table 4-7, “Job Execution Classes,” on page 167.

- The **Express** class is made up of all the jobs that have a higher priority than
“normal_jobs” in the preempt_prio scheduler parameter. The Express class is sorted first by applying the rules for preemption priority you set in the scheduler’s preempt_prio parameter, making preemption priority the first sort key. The second sort key is the formula or job_sort_key, depending on which is defined, or if neither is defined, job submission time.

Jobs are sorted into this class only when preemption is enabled. See section 4.8.33, “Using Preemption”, on page 228. Please note that execution priority classes are distinct from preemption levels, and are used for different purposes.

For example, if preempt_prio is the following:

preempt_prio: “express_queue, starving_jobs, normal_jobs”

The Express class contains all jobs that have preemption priority that is greater than that of normal jobs. In this example, the Express class is prioritized with top priority for express queue jobs, followed by starving jobs.

Since preemption levels are applied so that a job is put into the highest preemption level possible, in this example, all starving jobs end up in the Express class.

- The Starving class is sorted only according to the amount of time that counts toward starving for each job. You can use queue wait time or eligible time as starving time. Jobs are sorted into this class only when starving is enabled. See section 4.8.46, “Starving Jobs”, on page 279.

- The Reservation class is sorted within each reservation, according to the formula or job_sort_key, depending on which is defined. If neither is defined, jobs are sorted according to their submission time.

- The Suspended class is sorted according to the formula or job_sort_key, depending on which is defined. If neither is defined, jobs are sorted according to their submission time. All jobs in the Suspended class are sorted as if they are in a single queue.

- For Normal jobs, queue order is the first sort key, and, if defined, the formula, fairshare, or job sort key is the second sort key. If job_sort_formula, fair_share, and job_sort_key are not defined, the second sort key is job submission time.

One class is not used to divide or sort another. For example, if there are jobs in the Express class that are also starving or suspended, those factors are not used to sort the jobs in the Express class.

While the last sort key does sort jobs at the finest granularity, most of the work of sorting jobs is done at this level.
4.8.16.3.i  Precedence of Sort Method Used Within Class

If the formula is defined, it overrides fairshare and the job sort key. If fair share is defined, it overrides the job sort key. If none are defined, jobs are ordered by their arrival time in the queue.

For the job sorting formula, see section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.

For fairshare, see section 4.8.18, “Using Fairshare”, on page 170.

For sorting jobs on a key, see section 4.8.43, “Sorting Jobs on a Key”, on page 275.

4.8.16.4  Execution Priority Caveats

• Limits are not taken into account when prioritizing jobs for execution. Limits are checked only after setting priority, when selecting a job to run. The only exception is in the Express class, where soft limits may be taken into account, because execution priority for Express class jobs is calculated using preemption priority. For details, see section 4.8.33, “Using Preemption”, on page 228.

• When you issue “qrun <job ID>”, without the -H option, the selected job has execution priority between Reservation and Express.

• Jobs are sorted into the Express class is only when preemption is enabled. Similarly, jobs are sorted into the Starving class only when starving is enabled.

4.8.17  Express Queues

You can use express queues as tools to manage job execution and preemption priority:

• You can set up execution priority levels that include jobs in express queues. For information on configuring job priorities in the scheduler, see section 4.8.16, “Calculating Job Execution Priority”, on page 166.

• You can set up preemption levels that include jobs in express queues. For information on preemption, see section 4.8.33, “Using Preemption”, on page 228.

An express queue is a queue whose priority is high enough to qualify as an express queue; the default for qualification is 150, but this can be set using the preempt_queue_prio scheduler parameter. For information on configuring express queues, see section 2.2.5.3.i, “Express Queues”, on page 24.

The term “express” is also used in calculating execution priority to mean all jobs that have a preemption level greater than that of the normal_jobs level.
4.8.18 Using Fairshare

Fairshare provides a way to enforce a site's resource usage policy. It is a method for ordering the start times of jobs based on two things: how a site's resources are apportioned, and the resource usage history of site members. Fairshare ensures that jobs are run in the order of how deserving they are. The scheduler performs the fairshare calculations each scheduling cycle. If fairshare is enabled, all jobs have fairshare applied to them and there is no exemption from fairshare.

The administrator can employ basic fairshare behavior, or can apply a policy of the desired complexity.

The fair_share parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

4.8.18.1 Outline of How Fairshare Works

The owner of a PBS job can be defined for fairshare purposes to be a user, a group, the job’s accounting string, etc. For example, you can define owners to be groups, and can explicitly set each group’s relationship to all the other groups by using the tree structure. You can define one group to be part of a larger department.

The usage of exactly one resource is tracked for all job owners. So if you defined job owners to be groups, and you defined cput to be the resource that is tracked, then only the cput usage of groups is considered. PBS tries to ensure that each owner gets the amount of resources that you have set for it.

If you don’t explicitly list an owner, it will fall into the “unknown” catchall. All owners in “unknown” get the same resource allotment.

4.8.18.2 The Fairshare Tree

Fairshare uses a tree structure, where each vertex in the tree represents some set of job owners and is assigned usage shares. Shares are used to apportion the site’s resources. The default tree always has a root vertex and an unknown vertex. The default behavior of fairshare is to give all users the same amount of the resource being tracked. In order to apportion a site’s resources according to a policy other than equal shares for each user, the administrator creates a fairshare tree to reflect that policy. To do this, the administrator edits the file PBS_HOME/sched_priv/resource_group, which describes the fairshare tree.
4.8.18.3 Enabling Basic Fairshare

If the default fairshare behavior is enabled, all users with queued jobs will get an equal share of CPU time. The root vertex of the tree will have one child, the unknown vertex. All users will be put under the unknown vertex, and appear as children of the unknown vertex.

Basic fairshare is enabled by doing the following:

- In `PBS_HOME/sched_priv/sched_config`, set the scheduler configuration parameter `fair_share` to `True`.
- Uncomment the `unknown_shares` setting so that it is set to `unknown_shares: 10`.
- Specify how you want fairshare to work with primetime and non-primetime. If you want separate behavior for primetime and non-primetime, list the `fair_share` parameter twice, once for each time slot. The default is both. For example:

```
fair_share True prime
fair_share False non_prime
```

Note that a variant of basic fairshare has all users listed in the tree as children of root. Each user can be assigned a different number of shares. This must be explicitly created by the administrator.

4.8.18.4 Using Fairshare to Enforce Policy

The administrator sets up a hierarchical tree structure made up of interior vertices and leaves. Interior vertices are `departments`, which can contain both departments and leaves. Leaves are for `fairshare entities`, defined by setting `fairshare_entity` to one of the following: `euser`, `egroup`, `egroup:euser`, `Account_Name`, or `queue`. Apportioning of resources for the site is among these entities. These entities' usage of the designated resource is used in determining the start times of the jobs associated with them. All fairshare entities must be the same type. If you wish to have a user appear in more than one department, you can use `egroup:euser` to distinguish between that user's different resource allotments.

<table>
<thead>
<tr>
<th>Table 4-8: Using Fairshare Entities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keyword</strong></td>
</tr>
<tr>
<td><code>euser</code></td>
</tr>
</tbody>
</table>

PBS Professional 12.2 Administrator’s Guide 171
4.8.18.4.i Shares in the Tree

The administrator assigns shares to each vertex in the tree. The actual number of shares given to a vertex or assigned in the tree is not important. What is important is the ratio of shares among each set of sibling vertices. Competition for resources is between siblings only. The sibling with the most shares gets the most resources.

4.8.18.4.ii Shares Among Unknown Entities

The root vertex always has a child called unknown. Any entity not listed in PBS_HOME/sched_priv/resource_group will be made a child of unknown, designating the entity as unknown. The shares used by unknown entities are controlled by two parameters in PBS_HOME/sched_priv/sched_config: unknown_shares and fairshare_enforce_no_shares.

The parameter unknown_shares controls how many shares are assigned to the unknown vertex. The default sched_config file contains this line:

```bash
#unknown_shares 10
```

If you leave unknown_shares commented out, the unknown vertex will have 0 shares. If you simply remove the "#", the unknown vertex's shares default to 10. The children of the unknown vertex have equal amounts of the shares assigned to the unknown vertex.

The parameter fairshare_enforce_no_shares controls whether an entity without any shares can run jobs. If fairshare_enforce_no_shares is True, then entities without shares cannot run jobs. If it is set to False, entities without any shares can run jobs, but only when no other entities' jobs are available to run.

## Table 4-8: Using Fairshare Entities

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Fairshare Entities</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>egroup</td>
<td>OS group name</td>
<td>Groups as a whole are allotted shares of the resource being tracked.</td>
</tr>
<tr>
<td>egroup:euser</td>
<td>Combinations of username and group name</td>
<td>Useful when a user is a member of more than one group, and needs to use a different allotment in each group.</td>
</tr>
<tr>
<td>Account_Name</td>
<td>Account IDs</td>
<td>Shares are allotted by account string (Account_Name job attribute).</td>
</tr>
<tr>
<td>queues</td>
<td>Queues</td>
<td>Shares are allotted between queues.</td>
</tr>
</tbody>
</table>

egroup

OS group name

Groups as a whole are allotted shares of the resource being tracked.

egroup:euser

Combinations of username and group name

Useful when a user is a member of more than one group, and needs to use a different allotment in each group.

Account_Name

Account IDs

Shares are allotted by account string (Account_Name job attribute).

queues

Queues

Shares are allotted between queues.
4.8.18.4.iii Format for Describing the Tree

The file describing the fairshare tree contains four columns to describe the vertices in the tree. Here is the format for the columns:

<Vertex name>  <vertex fairshare ID>  <parent of vertex>  <#shares>

The columns are for a vertex's name, its fairshare ID, the name of its parent vertex, and the number of shares assigned to that vertex. Vertex names and IDs must be unique. Vertex IDs are integers. The top row in resource_group contains information for the first vertex, rather than column labels.

Neither the root vertex nor the unknown vertex is described in PBS_HOME/sched_priv/resource_group. They are always added automatically. Parent vertices must be listed before their children.

For example, we have a tree with two top-level departments, Math and Phys. Under Math are the users Bob and Tom as well as the department Applied. Under Applied are the users Mary and Sally. Under Phys are the users John and Joe. Our PBS_HOME/sched_priv/resource_group looks like this:

```
Math           100     root  30
Phys           200     root  20
Applied        110     Math  20
Bob            101     Math  20
Tom            102     Math  10
Mary           111     Applied 1
Sally          112     Applied 2
John           201     Phys  2
Joe            202     Phys  2
```

If you wish to use egroupeuser as your entity, and Bob to be in two UNIX/Windows groups pbsgroup1 and pbsgroup2, and Tom to be in two groups pbsgroup2 and pbsgroup3:

```
Math           100     root  30
Phys           200     root  20
Applied        110     Math  20
pbsgroup1:Bob  101     Phys  20
pbsgroup2:Bob  102     Math  20
pbsgroup2:Tom  103     Math  10
pbsgroup3:Tom  104     Applied 10
```
A user's egroup, unless otherwise specified, will default to their primary UNIX/Windows group. When a user submits a job using `-Wgroup_list=<group>`, the job's egroup will be `<group>`. For example, user Bob is in pbsgroup1 and pbsgroup2. Bob uses "qsub -Wgroup_list= pbsgroup1" to submit a job that will be charged to pbsgroup1, and "qsub -Wgroup_list=pbsgroup2" to submit a job that will be charged to pbsgroup2.

The first and third fields are alphanumeric. The second and fourth fields are numeric. Fields can be separated by spaces and tabs.

### 4.8.18.4.iv Computing How Much Each Vertex Deserves

How much resource usage each entity deserves is its portion of all the shares in the tree, divided by its past and current resource usage.

A vertex's portion of all the shares in the tree is called *tree percentage*. It is computed for all of the vertices in the tree. Since the leaves of the tree represent the entities among which resources are to be shared, their tree percentage sums to 100 percent.

The scheduler computes the tree percentage for the vertices this way:

1. For each internal vertex A:
   - sum the shares of its children;
2. For each child J of vertex A:
   - divide J's shares by the sum to normalize the shares;
   - multiply J's normalized shares by vertex A's tree percentage to find J's tree percentage.

### 4.8.18.5 Tracking Resource Usage

The administrator selects exactly one resource to be tracked for fairshare purposes by setting the scheduler configuration parameter `fairshare_usage_res` in `PBS_HOME/sched_priv/sched_config`.

Fairshare is effective only when the resource used is a time-based resource. The time-based resources are `cput` and the exact string "ncpus*walltime" which multiplies the number of CPUs requested by the `walltime` in seconds. The default for the tracked resource is `cput`, CPU time.

To use CPU time as the resource to be tracked, put this line in `PBS_HOME/sched_priv/sched_config`:

```
fairshare_usage_res: cput
```
To use `ncpus` multiplied by `walltime` as the resource to be tracked, put this line in

```
PBS_HOME/sched_priv/sched_config:
```

```
fairshare_usage_res: ncpus*walltime
```

An entity's usage always starts at 1. Resource usage tracking begins when the scheduler is started.

Each entity's current usage of the designated resource is combined with its previous usage. Each scheduler cycle, the scheduler adds the usage increment between this cycle and the previous cycle to its sum for the entity. Each entity's usage is *decayed*, or cut in half periodically, at the interval set in the `half_life` parameter in `PBS_HOME/sched_priv/sched_config`. This interval defaults to 24 hours.

This means that an entity with a lot of current or recent usage will have low priority for starting jobs, but if the entity cuts resource usage, its priority will go back up after a few decay cycles.

A static resource will not change its usage from one cycle to the next. If you use a static resource such as `ncpus`, the amount being tracked will not change during the lifetime of the job; it will only be added once when the job starts.

Note that if a job ends between two scheduling cycles, its resource usage for the time between previous scheduling cycle and the end of the job will not be recorded. The scheduler's default cycle interval is 10 minutes. The scheduling cycle can be adjusted via the `qmgr` command. Use

```
Qmgr: set server scheduler_iteration=<new value>
```

### 4.8.18.6 Finding the Most Deserving Entity

The most deserving entity is found by starting at the root of the tree, comparing its immediate children, finding the most deserving, then looking among that vertex's children for the most deserving child. This continues until a leaf is found. In a set of siblings, the most deserving vertex will be the vertex with the lowest ratio of resource usage divided by tree percentage.

### 4.8.18.7 Choosing Which Job to Run

The job to be run next will be selected from the set of jobs belonging to the most deserving entity. The jobs belonging to the most deserving entity are sorted according to the methods the scheduler normally uses. This means that fairshare effectively becomes the primary sort key. If the most deserving job cannot run, then the next most is selected to run, and so forth. All of the most deserving entity’s jobs would be examined first, then those of the next most deserving entity, et cetera.
Chapter 4  

**Scheduling**

At each scheduling cycle, the scheduler attempts to run as many jobs as possible. It selects the most deserving job, runs it if it can, then recalculates to find the next most deserving job, runs it if it can, and so on.

When the scheduler starts a job, all of the job's requested usage is added to the sum for the owner of the job for one scheduling cycle. The following cycle, the job's usage is set to the actual usage used between the first and second cycles. This prevents one entity from having all its jobs started and using up all of the resource in one scheduling cycle.
Files and Parameters Used in Fairshare

The following parameters from `PBS_HOME/sched_priv/sched_config` are used in fairshare:

Table 4-9: `PBS_HOME/sched_priv/sched_config` Parameters used in Fairshare

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>fair_share</td>
<td>[True/False] Enable or disable fair-share</td>
</tr>
<tr>
<td>fairshare_usage_res</td>
<td>Resource whose usage is to be tracked; default is cput</td>
</tr>
<tr>
<td>half_life</td>
<td>Decay time period; default is 24 hours</td>
</tr>
<tr>
<td>unknown_shares</td>
<td>Number of shares for unknown vertex; default 10, 0 if commented out</td>
</tr>
<tr>
<td>fairshare_entity</td>
<td>The kind of entity which is having fairshare applied to it. Leaves in the tree are this kind of entity. Default: euser.</td>
</tr>
<tr>
<td>fairshare_enforce_no_shares</td>
<td>If an entity has no shares, this controls whether it can run jobs. T: an entity with no shares cannot run jobs. F: an entity with no shares can only run jobs when no other jobs are available to run.</td>
</tr>
<tr>
<td>by_queue</td>
<td>If on, queues cannot be designated as fairshare entities, and fairshare will work queue by queue instead of on all jobs at once.</td>
</tr>
</tbody>
</table>

`PBS_HOME/sched_priv/resource_group`
Contains the description of the fairshare tree.

`PBS_HOME/sched_priv/usage`
Chapter 4  Scheduling

Contains the usage database.

_qmgr_

Used to set scheduler cycle frequency; default is 10 minutes.

_qmgr: set server scheduler_iteration=<new value>

job attributes

Used to track resource usage:

resources_used.<resource>

Default is cput.

4.8.18.9  Fairshare for Complex or Within Queues

You can use fairshare to compare all jobs in the complex, or within each queue. Fairshare with a queue means that the scheduler examines the jobs in a queue, and compares them to each other, to determine which job to start next.

To use fairshare for the entire complex, set the by_queue and round_robin scheduler configuration parameters to False.

To use fairshare within queues, set the by_queue scheduler parameter to True, and round_robin to False. If you want to examine queues in a particular order, prioritize the queues by setting each queue’s priority attribute, and set the sort_queues scheduler parameter to True. The sort_queues parameter is deprecated as of 12.2.

The scheduler configuration parameter by_queue in the file PBS_HOME/sched_priv/sched_config is set to True by default.

If by_queue is True, queues cannot be designated as fairshare entities.

4.8.18.10  Fairshare and Strict Ordering

Fairshare dynamically reorders the jobs with every scheduling cycle. Strict ordering is a rule that says we always run the next-most-deserving job. If there were no new jobs submitted, strict ordering could give you a snapshot of how the jobs would run for the next $n$ days.

Hence fairshare appears to break that. However, looked at from a dynamic standpoint, fairshare is another element in the strict order.
4.8.18.11  Fairshare and Entity Shares (Strict Priority)

If you enable entity shares (strict priority), you use the same fairshare tree that you would use for fairshare. Fairshare and entity shares (strict priority) are incompatible, so in order to use entity shares, you disable fairshare by setting `fair_share` to `False`. For how to configure entity shares, see section 4.8.14, “Sorting Jobs by Entity Shares (Was Strict Priority)”, on page 160.

4.8.18.12  Viewing and Managing Fairshare Data

The `pbsfs` command provides a command-line tool for viewing and managing some fairshare data. You can display the tree in tree form or in list form. You can print all information about an entity, or set an entity's usage to a new value. You can force an immediate decay of all the usage values in the tree. You can compare two fairshare entities. You can also remove all entities from the `unknown` department. This makes the tree easier to read. The tree can become unwieldy because entities not listed in the file `PBS_HOME/sched_priv/resource_group` all land in the `unknown` group.

The fairshare usage data is written to the file `PBS_HOME/sched_priv/usage`, whenever the scheduler has new usage data. The usage data is always up-to-date.

For more information on using the `pbsfs` command, see “pbsfs” on page 101 of the PBS Professional Reference Guide.

4.8.18.13  Using Fairshare in Job Execution Priority

Jobs are sorted as specified by the formula in `job_sort_formula`, if it exists, or by fairshare, if it is enabled and there is no formula, or if neither of those is used, by `job_sort_key`. The job sorting formula can use the value of `fair_share_perc`, the percentage of the fairshare tree for this job's entity. See section 4.8.16, “Calculating Job Execution Priority”, on page 166.

4.8.18.14  Using Fairshare in Job Preemption Priority

You can use the `fairshare` preemption level in determining job preemption priority. This level applies to jobs whose owners are over their fairshare allotment. See section 4.8.33, “Using Preemption”, on page 228.
4.8.18.15 Fairshare and Requeued Jobs

When a job is requeued, it normally retains its original place in its execution queue with its former priority. The job is usually the next job to be considered during scheduling, unless the relative priorities of the jobs in the queue have changed. This can happen when the job sorting formula assigns higher priority to another job, another higher-priority job is submitted after the requeued job started, this job’s owner has gone over their fairshare limit, etc.

4.8.18.16 Fairshare Caveats

- If the job sorting formula is defined, it overrides fairshare.
- Do not use fairshare with help_starving_jobs.
- We do not recommend using fairshare with strict_ordering, or with strict_ordering and backfill. The results may be non-intuitive. Fairshare will cause relative job priorities to change with each scheduling cycle. It is possible that a job from the same entity or group as the top job will be chosen as the filler job. The usage from the filler job will lower the priority of the most deserving, i.e. top, job. This could delay the execution of the top job.
- Do not use fairshare when using the fair_share_perc option to job_sort_key.
- Do not use static resources such as ncpus as the resource to track. The scheduler adds the incremental change in the tracked resource at each scheduling cycle, and a static resource will not change.
- The most deserving entity can change with every scheduling cycle, if each time a job is run, it changes usage sufficiently.
- Fairshare dynamically reorders the jobs with every scheduling cycle. Strict ordering is a rule that says we always run the next-most-deserving job. If there were no new jobs submitted, strict ordering could give you a snapshot of how the jobs would run for the next $n$ days. Hence fairshare appears to break that. However, looked at from a dynamic standpoint, fairshare is another element in the strict order.

4.8.19 FIFO Scheduling

With FIFO scheduling, PBS runs jobs in the order in which they are submitted. You can use FIFO order for all of the jobs in your complex, or you can go queue by queue, so that the jobs within each queue are considered in FIFO order.
4.8.19.1 Configuring Basic FIFO Scheduling

To configure basic FIFO scheduling, whether across the complex or queue by queue, set the following scheduler parameters to these values:

- `round_robin`: False ALL
- `job_sort_key`: (commented out)
- `fair_share`: False ALL
- `help_starving_jobs`: False ALL
- `backfill`: False ALL
- `job_sort_formula`: (unset)

4.8.19.2 FIFO for Entire Complex

To configure FIFO across your entire complex, do one of the following:

- Use only one execution queue
- Set the `by_queue` scheduler parameter to `False`

4.8.19.3 Queue by Queue FIFO

To configure FIFO for each queue separately, first decide how you want queues to be selected. You can set the order in which PBS chooses queues from which to run jobs, or you can allow the queues to be selected in an undefined way.

- To allow queues to be selected in an undefined way, set the `by_queue` scheduler parameter to `True`.
- To set the order in which queues are selected, do the following:
  - Specify a priority for each queue
  - Set the `sort_queues` scheduler parameter to `True`. The `sort_queues` parameter is `deprecated` as of 12.2.
  - Set the `by_queue` scheduler parameter to `True`

4.8.19.4 FIFO with Strict Ordering

If your jobs must run exactly in submission order, you can use strict ordering with FIFO scheduling. If you use strict ordering with FIFO scheduling, this means that when the job that is supposed to run next cannot run, no jobs can run. This can result in less throughput than you could otherwise achieve. To avoid that problem, you can use backfilling. See the following section.
Chapter 4  Scheduling

To use strict ordering with FIFO scheduling, use the following scheduler parameter settings in PBS_HOME/sched_priv/sched_config:

- strict_ordering: True ALL
- round_robin: False ALL
- job_sort_key: (commented out)
- fair_share False ALL
- help_starving_jobs False ALL
- backfill: False ALL
- job_sort_formula: (unset)

### 4.8.19.5 FIFO with Strict Ordering and Backfilling

If you want to run your jobs in submission order, except for backfilling around top jobs that are stuck, use the following:

- strict_ordering: True ALL
- round_robin: False ALL
- job_sort_key: (commented out)
- fair_share False ALL
- help_starving_jobs False ALL
- backfill: True ALL
- job_sort_formula: (unset)

### 4.8.20 Using a Formula for Computing Job Execution Priority

You can choose to use a formula by which to sort jobs at the finest-granularity level. The formula can only direct how jobs are sorted at the finest level of granularity. However, that is where most of the sorting work is done.

When the scheduler sorts jobs according to the formula, it computes a priority for each job. The priority computed for each job is the value produced by the formula. Jobs with a higher value get higher priority. See section 4.8.16.3, “Sorting Jobs Within Classes”, on page 167 for how the formula is used in setting job execution priority.

This formula will override both job_sort_key and fair_share for sorting jobs. If the job_sort_formula server attribute contains a formula, the scheduler will use it. If not, and fairshare is enabled, the scheduler computes job priorities according to fairshare. If neither the formula nor fairshare is defined, the scheduler uses job_sort_key.
Only one formula is used to prioritize all jobs. At each scheduling cycle, the formula is applied to all jobs, regardless of when they were submitted. If you change the formula, the new formula is applied to all jobs.

For example, if you submit some jobs, change the formula, then submit more jobs, the new formula is used for all of the jobs, during the next scheduling cycle.

You may find that the formula is most useful when you use it with custom resources inherited by or allocated to jobs. For example, you may want to route all jobs from a particular project to a queue where they inherit a specific value for a custom resource. Other jobs may end up at a different queue, where they inherit a different value, or they may inherit no value. You can then use this custom resource in the formula as a way to manage job priority. See section 11.2, “Allocating Resources to Jobs”, on page 821, and section 4.8.8, “Using Custom and Default Resources”, on page 133.

It may be helpful if these custom resources are invisible and unrequestable by users. See section 4.8.20.9, “Examples of Using Resource Permissions in Job Sorting Formula”, on page 189.

4.8.20.1 Using the Formula

Once you set job_sort_formula via qmgr, it takes effect with the following scheduling cycle. Variables are evaluated at the start of the scheduling cycle.

If an error is encountered while evaluating the formula, the formula evaluates to zero for that job, and the following message is logged at event class 0x0100:

“1234.mars;Formula evaluation for job had an error. Zero value will be used”

4.8.20.2 Configuring the Job Sorting Formula

You specify the formula in the server’s job_sort_formula attribute. To set the job_sort_formula attribute, use the qmgr command. When specifying the formula, be sure to follow the requirements for entering an attribute value via qmgr: strings containing whitespace, commas, or other special characters must be enclosed in single or double quotes. See “Attribute Values” on page 158 of the PBS Professional Reference Guide. Format:

```
Qmgr: s s job_sort_formula = "<formula>"
```

4.8.20.3 Requirements for Creating Formula

The job sorting formula must be created at the server host.
Chapter 4  Scheduling

Under UNIX/Linux, root privilege is required in order to operate on the job_sort_formula server attribute.

Under Windows, this must be done from the installation account. For domained environments, the installation account must be a local account that is a member of the local Administrators group on the local computer. For standalone environments, the installation account must be a local account that is a member of the local Administrators group on the local computer.

4.8.20.4 Format of Formula

The formula can be made up of any number of expressions, where expressions contain terms which are added, subtracted, multiplied, or divided. You can use parentheses, exponents, and unary + and - operators. All operators use standard mathematical precedence.

The formula can be any length.

The range for the formula is defined by the IEEE floating point standard for a double.

4.8.20.5 Units in Formula

The variables you can use in the formula have different units. Make sure that some terms do not overpower others, by normalizing them where necessary. Resources like ncpus are integers, size resources like mem are in kb, so 1gb is 1048576kb, and time-based resources are in seconds (e.g. walltime). Therefore, if you want a formula that combines memory and ncpus, you’ll have to account for the factor of 1024 difference in the units.
The following are the units for the supported built-in resources:

**Table 4-10: Job Sorting Formula Units**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Units</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time resources</td>
<td><em>Integer number of seconds</em></td>
<td>300</td>
</tr>
<tr>
<td>Memory</td>
<td><em>kb</em></td>
<td>1gb =&gt; 1048576kb</td>
</tr>
<tr>
<td>ncpus</td>
<td><em>Integer</em></td>
<td>8</td>
</tr>
</tbody>
</table>

Example 4-2: If you use ‘1 * ncpus + 1 * mem’, where mem=2mb, ncpus will have almost no effect on the formula result. However, if you use ‘1024 * ncpus + 1 * mem’, the scaled mem won’t overpower ncpus.

Example 4-3: You are using gb of mem:

```
Qmgr: s s job_sort_formula='1048576 * ncpus + 2 * mem'
```

Example 4-4: If you want to add days of walltime to queue priority, you might want to multiply the time by 0.0000115, equivalent to dividing by the number of seconds in a day:

```
Qmgr: s s job_sort_formula = '.0000115*walltime + queue_priority'
```

### 4.8.20.6 Formula Coefficients

The formula operates only on resources in the job’s Resource_List attribute. These are the numeric job-level resources, and may have been explicitly requested, inherited, or summed from consumable host-level resources. See section 5.9.2, “Resources Requested by Job”, on page 305.

This means that all variables and coefficients in the formula must be resources that were either requested by the job or were inherited from defaults at the server or queue. These variables and coefficients can be custom numeric resources inherited by the job from the server or queue, or they are long integers or floats.

You may need to create custom resources at the server or queue level to be used for formula coefficients. See section 4.8.8, “Using Custom and Default Resources”, on page 133.
The following table lists the terms that can be used in the formula:

**Table 4-11: Terms in Job Sorting Formula**

<table>
<thead>
<tr>
<th>Terms</th>
<th>Allowable Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constants</td>
<td>NUM or NUM.NUM</td>
</tr>
<tr>
<td>Attribute values</td>
<td></td>
</tr>
<tr>
<td>queue_priority</td>
<td>Value of priority attribute for queue in which job resides</td>
</tr>
<tr>
<td>job_priority</td>
<td>Value of the job’s priority attribute</td>
</tr>
<tr>
<td>fair_share_perc</td>
<td>Percentage of fairshare tree for this job’s entity</td>
</tr>
<tr>
<td>eligible_time</td>
<td>Amount of wait time job has accrued while waiting for resources</td>
</tr>
<tr>
<td>Resources</td>
<td></td>
</tr>
<tr>
<td>ncpus</td>
<td></td>
</tr>
<tr>
<td>mem</td>
<td></td>
</tr>
<tr>
<td>walltime</td>
<td></td>
</tr>
<tr>
<td>cput</td>
<td></td>
</tr>
</tbody>
</table>
| Custom numeric job-wide resources | Uses the amount requested, not the amount used. Must be of type long, float, or size. See section 5.14.2.8, “Custom Resource Values”, on page 331.
4.8.20.8 Examples of Using the Job Sorting Formula

Examples of formulas:

Example 4-5: \(10 \times \text{ncpus} + 0.01 \times \text{walltime} + A \times \text{mem}\)

Here, “A” is a custom resource.

Example 4-6: \(\text{ncpus} + 0.0001 \times \text{mem}\)

Example 4-7: To change the formula on a job-by-job basis, alter the value of a resource in the job’s Resource_List.<resource>. So if the formula is \(A \times \text{queue\_priority} + B \times \text{job\_priority} + C \times \text{ncpus} + D \times \text{walltime}\), where A-D are custom numeric resources.
Chapter 4

Scheduling

These resources can have a default value via `resources_default.A` ...
`resources_default.D`. You can change the value of a job’s resource through `qalter`.

Example 4-8: ncpus*mem

Example 4-9: Set via qmgr:

```bash
qmgr -c 'set server job_sort_formula= 5*ncpus+0.05*walltime'
```

Following this, the output from `qmgr -c 'print server'` will look like

```
set server job_sort_formula="5*ncpus+0.05*walltime"
```

Example 4-10:

```bash
Qmgr: s s job_sort_formula=ncpus
```

Example 4-11:

```bash
Qmgr: s s job_sort_formula=’queue_priority + ncpus’
```

Example 4-12:

```bash
Qmgr: s s job_sort_formula=’5*job_priority + 10*queue_priority’
```

Example 4-13: Sort jobs using the value of `ncpus` x `walltime`:

Formula expression: “ncpus * walltime”

Submit these jobs:

Job 1: ncpus=2 walltime=01:00:00 -> 2*60s = 120
Job 2: ncpus=1 walltime=03:00:00 -> 1*180s = 180
Job 3: ncpus=5 walltime=01:00:00 -> 5*60s = 300

The scheduler logs the following:

Job ;1.host1;Formula Evaluation = 120
Job ;2.host1;Formula Evaluation = 180
Job; 3.host1;Formula Evaluation = 300

The jobs are sorted in the following order:

Job 3
Job 2
Job 1
4.8.20.9 Examples of Using Resource Permissions in Job Sorting Formula

See section 5.14.2.6, “Resource Permission Flags”, on page 328 for information on using resource permissions.

Example 4-14: You may want to create per-job coefficients in your job sorting formula which are set by system defaults and which cannot be viewed, requested or modified by the user. To do this, you create custom resources for the formula coefficients, and make them invisible to users. In this example, A, B, C and D are the coefficients. You then use them in your formula:

\[ A \times \text{Queue Priority} + B \times \text{Job Class Priority} + C \times \text{CPUs} + D \times \text{Queue Wait Time} \]

Example 4-15: You may need to change the priority of a specific job, for example, have one job or a set of jobs run next. In this case, you can define a custom resource for a special job priority. If you do not want users to be able to change this priority, set the resource permission flag for the resource to r. If you do not want users to be able to see the priority, set its resource permission flag to i. For the job or jobs that you wish to give top priority, use qalter to set the special resource to a value much larger than any formula outcome.

Example 4-16: To use a special priority:

\[ \text{sched\_priority} = W_{\text{prio}} \times \text{wait\_secs} + P_{\text{prio}} \times \text{priority} + \ldots + \text{special\_priority} \]

Here, special\_priority is very large.

4.8.20.10 Caveats and Error Messages

- It is invalid to set both job\_sort\_formula and job\_sort\_key at the same time. If they are both set, job\_sort\_key is ignored and the following error message is logged:
  “Job sorting formula and job\_sort\_key are incompatible. The job sorting formula will be used.”

- If the formula overflows or underflows the sorting behavior is undefined.

- If you set the formula to an invalid formula, qmgr will reject it, with one of the following error messages:
  “Invalid Formula Format”
  “Formula contains invalid keyword”
  “Formula contains a resource of an invalid type”

- If an error is encountered while evaluating the formula, the formula evaluates to zero for
that job, and the following message is logged at event class 0x0100:
“1234.mars;Formula evaluation for job had an error. Zero value will be used”

- The job sorting formula must be set via qmgr at the server host.
- When a job is moved to a new server or queue, it will inherit new default resources from that server or queue. If it is moved to a new server, it will be prioritized according to the formula on that server, if one exists.
- If the job is moved to another server through peer scheduling and the pulling server uses queue priority in its job sorting formula, the queue priority used in the formula will be that of the queue to which the job is moved.
- If you are using FIFO scheduling, the job_sort_formula server attribute must be unset.
- If you are using eligible time in the formula, and eligible_time_enable is False, each job’s eligible time evaluates to zero in the formula.
- If a job is requeued, and you are using the formula, the job may lose its place, because various factors may affect the job’s priority. For example, a higher-priority job may be submitted between the time the job is requeued and the time it would have run, or another job’s priority may be increased due to changes in which jobs are running or waiting.
- If the formula is configured, it is in force during both primetime and non-primetime.
- If the job sorting formula is defined, it overrides fairshare.

### 4.8.20.11 Logging

For each job, the evaluated formula answer is logged at the highest log event class (0x0400):
“Formula Evaluation = <answer>”

### 4.8.21 Gating Jobs at Server or Queue

You can set resource limits at the server and queues so that jobs must conform to the limits in order to be admitted. This way, you can reject jobs that request more of a resource than the complex or a queue can supply.

You can also force jobs into specific queues where they will inherit the desired values for unrequested or custom resources. You can then use these resources to manage jobs, for example by using the resources in the job sorting formula or to route jobs to particular vnodes.

You can either force users to submit their jobs to specific queues, or you can have users submit jobs to routing queues, and then route the jobs to the desired queues.

For information on using resources for gating, see section 5.13, “Using Resources to Restrict Server, Queue Access”, on page 318.
For a description of which resources can be used for gating, see section 2.2.6.4.iii, “Resources Used for Routing and Admittance”, on page 27.

For how queue resource limits are applied to jobs, see section 2.2.6.4.i, “How Queue and Server Limits Are Applied, Except Running Time”, on page 26.

For how routing queues work, see section 2.2.6, “Routing Queues”, on page 24.

For how to route jobs to particular vnodes, see section 4.8.2, “Associating Vnodes with Queues”, on page 121.

For how to use resources in the job sorting formula, see section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.

### 4.8.21.1 Gating Caveats

- For most resources, if the job does not request the resource, and no server or queue defaults are set, the job inherits the maximum gating value for the resource. See section 5.9.3.7, “Using Gating Values As Defaults”, on page 308.

- For shrink-to-fit jobs, if a walltime limit is specified:
  - Both min_walltime and max_walltime must be greater than or equal to resources_min.walltime.
  - Both min_walltime and max_walltime must be less than or equal to resources_max.walltime.

### 4.8.22 Managing Application Licenses

PBS does not check application licenses out from the license server. PBS has no direct control over application licenses. However, you can have the scheduler use a dynamic resource to track application license use. This way, the scheduler knows how many application licenses are available, and how many have been checked out. For how to configure dynamic resources to represent application licenses, see section 5.14.7, “Supplying Application Licenses”, on page 346.

Unfortunately, some jobs or applications don’t check out all of the application licenses they use until they have been running for some time. For example, job J1, which requests licenses, starts running, but doesn’t check them out for a few minutes. Next, the scheduler considers job J2, which also requests licenses. The scheduler runs its query for the number of available licenses, and the query returns with a sufficient number of licenses to run J2, so the scheduler starts J2. Shortly afterward, J1 checks out licenses, leaving too few to run J2.
Chapter 4  Scheduling

It might appear that you could track the number of application licenses being used with a 
static integer PBS resource, and force jobs requesting application licenses to request this 
resource as well, but there is a drawback: if a job that has requested this resource is sus-
pended, its static resources are released, but its application licenses are not. In this case you 
could end up with a deceptively high number for available licenses.

You can limit the number of jobs that request application licenses, if you know how many 
jobs can run at one time:

• Create a custom server-level consumable integer resource to represent these jobs. See 

• Use `qmgr` to set `resources_available.<job limit>` at the server to the number of jobs that 
can run at one time.

• Force all jobs requesting the application to request one of these. See section 11.2, “Allo-
cating Resources to Jobs”, on page 821.

4.8.23 Limits on Per-job Resource Usage

You can specify how much of each resource any job is allowed to request, at the server and 
queue level. The server and queues each have per-job limit attributes. The 
`resources_min.<resource>` and `resources_max.<resource>` server and queue attributes are 
limits on what each individual job may request.

You cannot set `resources_min` or `resources_max` limits on `min_walltime` or `max_walltime`.

See section 5.15.3, “Placing Resource Limits on Jobs”, on page 391, and section 5.13, “Using 
Resources to Restrict Server, Queue Access”, on page 318.

4.8.24 Limits on Project, User, and Group Jobs

You can manage the number of jobs being run by users or groups, and the number of jobs 
being run in projects, at the server or queue level. For example, you can limit the number of 
jobs enqueued in queue QueueA by any one group to 30, and by any single user to 5.

See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & 
Queues”, on page 367.

4.8.25 Limits on Project, User, and Group Resource Usage

You can manage the total amount of each resource that is used by projects, users, or groups, at 
the server or queue level. For example, you can manage how much memory is being used by 
jobs in queue QueueA.

4.8.26 Limits on Jobs at Vnodes

You can set limits on the number of jobs that can be run at each vnode by users, by groups, or overall. See section 5.15.2, “Limiting Number of Jobs at Vnode”, on page 390.

4.8.27 Using Load Balancing

PBS can track the load on each execution host, running new jobs on the host according to the load on the host. You can specify that PBS does this for all machines in the complex. This is somewhat different behavior from that used for managing the load on vnodes; when managing load levels on vnodes, the scheduler only pays attention to the state of the vnode, and does not calculate whether a job would put the vnode over its load limit. Managing load levels on vnodes does not require load balancing to be turned on. See section 9.4.5, “Managing Load Levels on Vnodes”, on page 738.

You use the load_balancing scheduler parameter to control whether PBS tracks the load on each host.

The load_balancing parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

4.8.27.1 How Load Average is Computed

When load balancing is on, the scheduler queries each MoM once each scheduling cycle for the MoM’s load. MoM checks the load average on her host every 10 seconds.

The load used by MoM is the following:

- On UNIX/Linux, it is the raw one-minute averaged “loadave” returned by the operating system
- On Windows, it is based on the processor queue length

When a new load is added to a vnode, the load average increases slowly over time, so that more jobs than you want may be started at first. Eventually, the load average matches the actual load. If this is above the limit, PBS won’t start any more jobs on that vnode. As jobs terminate, the load average slowly moves down, and it takes time before the vnode is chosen for new jobs.

Consult your OS documentation to determine load values that make sense.

MoM sets the load only on the natural vnode, so it is the same for all vnodes on a multi-vnode machine.
4.8.27.2  How PBS Uses Load Information

When choosing vnodes for a job, the scheduler considers the load on the vnode in addition to whether the vnode can supply the resources specified in the job’s Resource_List attribute.

PBS estimates that a 1-CPU job will produce one CPU’s worth of load. This means that if you have a 2-CPU machine whose load is zero, PBS will put two 1-CPU jobs, or one 2-CPU job, on that machine.

When using load balancing, if a vnode has gone above $\text{max\_load}$, PBS does not run new jobs on the vnode until the load drops below $\text{ideal\_load}$.

MoM sets the vnode’s state according to its load. When a vnode’s load goes above $\text{max\_load}$, MoM marks the vnode busy. When the load drops below $\text{ideal\_load}$, MoM marks the vnode free. When a vnode’s state changes, for example from free to busy, MoM informs the server.

When using load balancing, PBS does not run new jobs on vnodes under the following conditions:

- Vnodes that are marked busy
- Vnodes whose resources, such as ncpus, are already fully allocated
- Vnodes that are above $\text{max\_load}$
- Vnodes where running the job would cause the load to go above $\text{max\_load}$

4.8.27.3  When to Use Load Balancing

When using load balancing (meaning the load_balancing scheduler parameter is True), the only changes to behavior are the following:

- The scheduler won’t place a job on a vnode whose load is above $\text{max\_load}$
- The scheduler won’t place a job on a vnode where that job would put the load above $\text{max\_load}$

Load balancing is useful when you want to oversubscribe CPUs, managing job placement by load instead. This can help when you want to run lots of jobs where each job will need only some CPU time, and the average load on the machine will be reasonable.

4.8.27.4  Suspending Jobs on Overloaded Vnodes

You can specify that MoM should suspend jobs when the load goes above $\text{max\_load}$, by adding the suspend argument to the $\text{max\_load}$ parameter. See section “$\text{max\_load <load> [suspend]}$”, on page 196. In this case, MoM suspends all jobs on the vnode until the load drops below $\text{ideal\_load}$, then resumes them. This option is useful only when the source of
the load is not strictly PBS jobs. This option is not recommended when the load is due solely to PBS jobs, because it can lead to the vnode cycling back and forth between becoming overloaded, being marked busy, suspending all jobs, being marked free, then starting all jobs, becoming overloaded, and so on.

4.8.27.5 Configuring Load Balancing

If you want to oversubscribe CPUs, set the value of ncpus on the vnode to the desired higher value.

We recommend setting the value of $max_load to a slightly higher value than the desired load, for example .25 + ncpus. Otherwise, the scheduler will not schedule jobs onto the last CPU, because it thinks a 1-CPU job will raise the load by 1, and the machine probably registers a load above zero.

To configure load balancing, perform the following steps:

1. Turn on load balancing by setting the load_balancing scheduler parameter to True:
   
   ```
   load_balancing: True ALL
   ```

2. Choose whether you want load balancing during primetime, non-primetime, or all. If you want separate behavior for primetime and non-primetime, specify each separately. The default is both. Example of separate behavior:

   ```
   load_balancing True prime
   load_balancing False non_prime
   ```

3. Set the ideal and maximum desired load for each execution host, by specifying values for $ideal_load and $max_load in each execution host’s MoM configuration file:

   ```
   $ideal_load <value at which to start new jobs>
   $max_load   <value at which to cease starting jobs>
   ```

4. Set each host’s resources_available.ncpus to the maximum number of CPUs you wish to allocate on that host. Follow the recommendations in section 3.5.2, “Choosing Configuration Method”, on page 52.

4.8.27.6 Load Balancing Caveats and Recommendations

- When setting ncpus and $max_load, consider the ratio between the two. PBS won’t allocate more than the value of resources_available.ncpus, so you can use this value to keep the load average from getting too high.
- Make sure that load balancing does not interfere with communications. Please read section 9.4.5, “Managing Load Levels on Vnodes”, on page 738.
- Load balancing is incompatible with sorting vnodes on a key (node_sort_key) when
Chapter 4  

Scheduling

sorting on a resource using the “unused” or “assigned” parameters. Load balancing will be disabled. See section 4.8.48, “Sorting Vnodes on a Key”, on page 283.

- You can use load balancing with SMP cluster distribution, but `smp_cluster_dist` will behave as if it is set to `pack`. See section 4.8.42, “SMP Cluster Distribution”, on page 273.

- We recommend setting the value of `$max_load` to a slightly higher value than the desired load, for example `.25 + ncpus`. Otherwise, the scheduler will not schedule jobs onto the last CPU, because it thinks a 1-CPU job will raise the load by 1, and the machine probably registers a load above zero.

- If you are using cycle harvesting via load balancing, make sure your load balancing settings do not interfere with cycle harvesting. Be careful with the settings for `$ideal_load` and `$max_load`. You want to make sure that when the workstation owner is using the machine, the load on the machine triggers MoM to report being busy, and that PBS does not start any new jobs while the user is working. Please read section 4.8.9.6, “Cycle Harvesting Based on Load Average”, on page 147.

- Using load balancing with multi-vnoded machines is not supported. MoM sets the load average only on the natural vnode, so all vnodes on a multi-vnoded machine are given the same value regardless of their actual load.

- It is not recommended to specify that MoM should suspend jobs when the load goes above `$max_load`. See section 4.8.27.4, “Suspending Jobs on Overloaded Vnodes”, on page 194.

- If you configure both placement sets and load balancing, the net effect is that vnodes that are over their load limit will be removed from consideration.

### 4.8.27.7 Parameters Affecting Load Balancing

#### `$ideal_load <load>`
MoM parameter. Defines the load below which the vnode is not considered to be busy. Used with the `$max_load` parameter.

Example:

```
$ideal_load 1.8
```

Format: `Float`

No default

#### `$max_load <load> [suspend]`
MoM parameter. Defines the load above which the vnode is considered to be busy. Used with the `$ideal_load` parameter.
Scheduling

If the optional suspend argument is specified, PBS suspends jobs running on the vnode when the load average exceeds $max_load$, regardless of the source of the load (PBS and/or logged-in users).

Example:

$max_load 3.5$

Format: Float

Default: number of CPUs

load_balancing <T|F> [time slot specification]
Scheduler parameter. When set to True, the scheduler takes into account the load average on vnodes as well as the resources listed in the resources: line in sched_config. See “load_balancing” on page 287 of the PBS Professional Reference Guide.

Format: Boolean

Default: False all

4.8.28 Matching Jobs to Resources

The scheduler places each job where the resources requested by the job are available. The scheduler handles built-in and custom resources the same way. For a complete description of PBS resources, see Chapter 5, "PBS Resources", on page 287.

4.8.28.1 Scheduling on Consumable Resources

The scheduler constrains the use of a resource to the value that is set for that resource in resources_available.<resource>. For a consumable resource, the scheduler won’t place more demand on the resource than is available. For example, if a vnode has resources_available.ncpus set to 4, the scheduler will place jobs requesting up to a total of 4 CPUs on that vnode, but no more.

The scheduler computes how much of a resource is available by subtracting the total of resources_assigned.<resource> for all running jobs and started reservations from resources_available.<resource>.

4.8.28.2 Scheduling on Non-Consumable Resources

For non-consumable resources such as arch or host, the scheduler matches the value requested by a job with the value at one or more vnodes. Matching a job this way does not change whether or not other jobs can be matched as well; non-consumable resources are not used up by jobs, and therefore have no limits.
4.8.28.3 Scheduling on Dynamic Resources

At each scheduling cycle, the scheduler queries each dynamic resource. If a dynamic resource is not under the control of PBS, jobs requesting it may run in an unpredictable fashion.

4.8.28.4 Scheduling on the walltime Resource

The scheduler looks at each job in priority order, and tries to run the job. The scheduler checks whether there is an open time slot on the requested resources that is at least as long as the job’s walltime. If there is, the scheduler runs the job.

PBS examines each shrink-to-fit job when it gets to it, and looks for a time slot whose length is between the job’s min_walltime and max_walltime. If the job can fit somewhere, PBS sets the job’s walltime to a duration that fits the time slot, and runs the job. For more information about shrink-to-fit jobs, see section 4.8.41, “Using Shrink-to-fit Jobs”, on page 263.

4.8.28.4.i Caveats for Scheduling on walltime

Do not set values for resources such as walltime at the server or a queue, because the scheduler will not allocate more than the specified value. This means that if you set resources_available.walltime at the server to 10:00:00, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.

4.8.28.5 Unrequestable or Invisible Resources

You can define custom resources that are invisible to and unrequestable by users, or simply unrequestable by users. The scheduler treats these resources the same as visible, requestable resources. See section 5.14.2.6, “Resource Permission Flags”, on page 328.

4.8.28.6 Enforcing Scheduling on Resources

The scheduler chooses which resources to schedule on according to the following rules:

- The scheduler always schedules jobs based on the availability of the following vnode-level resources:
  - vnode
  - host
  - Any Boolean resource

- The scheduler will schedule jobs based on the availability of other resources only if those resources are listed in the “resources:” line in PBS_HOME/sched_priv/
sched_config. Some resources are automatically added to this line. You can add resources to this line. The following resources are automatically added to this line:

- aoe
- arch
- host
- mem
- ncpus
- netwins
- vnode

### 4.8.28.7 Matching Unset Resources

When job resource requests are being matched with available resources, unset resources are treated as follows:

- A numerical resource that is unset on a host is treated as if it were `zero`
- An unset resource on the server or queue is treated as if it were infinite
- An unset string cannot be matched
- An unset Boolean resource is treated as if it were set to `False`.
- The resources `ompthreads`, `mpiprocs`, and `nodes` are ignored for unset resource matching.

The following table shows how a resource request will or won’t match an unset resource at the host level.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Unset Resource</th>
<th>Matching Request Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td><code>False</code></td>
<td><code>False</code></td>
</tr>
<tr>
<td>float</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>long</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>size</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>string</td>
<td>Never matches</td>
<td></td>
</tr>
<tr>
<td>string array</td>
<td>Never matches</td>
<td></td>
</tr>
</tbody>
</table>
4.8.28.7.i When Dynamic Resource Script Fails

If a server dynamic resource script fails, the scheduler uses the value of resources_available.<resource>. If this was never set, it is treated as an unset resource, described above.

If a host-level dynamic resource script fails, the scheduler treats the resource as if its value is zero.

4.8.28.7.ii Backward Compatibility of Unset Resources

To preserve backward compatibility, you can set the server’s resource_unset_infinite attribute with a list of host-level resources that will behave as if they are infinite when they are unset. See “resource_unset_infinite” on page 293 of the PBS Professional Reference Guide for information on resource_unset_infinite.

4.8.28.8 Resource Scheduling Caveats

• Do not set values for resources such as walltime at the server or a queue, because the scheduler will not allocate more than the specified value. This means that if you set resources_available.walltime at the server to 10:00:00, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.

• Jobs may be placed on different vnodes from those where they would have run in earlier versions of PBS. This is because a job’s resource request will no longer match the same resources on the server, queues and vnodes.

• Beware of application license race conditions. If two jobs require the same application license, the first job may be started, but may not get around to using the license before the second job is started and uses the license. The first job must then wait until the license is available, taking up resources. The scheduler cannot avoid this problem.

4.8.29 Node Grouping

The term “node grouping” has been superseded by the term “placement sets”. Vnodes were originally grouped according to the value of one resource, so for example all vnodes with a value of linux for arch were grouped together, and all vnodes with a value of solaris for arch...
were in a separate group. We use placement sets now because this means grouping vnodes according to the value of one or more resources. See section 4.8.32, “Placement Sets”, on page 211.

### 4.8.29.1 Configuring Old-style Node Grouping

Configuring old-style node grouping means that you configure the simplest possible placement sets. In order to have the same behavior as in the old node grouping, group on a single resource. If this resource is a string array, it should only have one value on each vnode. This way, each vnode will be in only one node group.

You enable node grouping by setting the server’s `node_group_enable` attribute to `True`.

You can configure one set of vnode groups for the entire complex by setting the server’s `node_group_key` attribute to a resource name.

You can configure node grouping separately for each queue by setting that queue’s `node_group_key` attribute to a resource name.

### 4.8.30 Overrides

You can use various overrides to change how one or more jobs run.

#### 4.8.30.1 Run a Job Manually

You can tell PBS to run a job now, and you can optionally specify where to run it. You run a job manually using the `qrun` command.

The `-H` option to the `qrun` command makes an important difference:

```
qrun
```

When preemption is enabled, the scheduler preempts other jobs in order to run this job. Running a job via `qrun` gives the job higher preemption priority than any other class of job, except for reservation jobs.

When preemption is not enabled, the scheduler runs the job only if enough resources are available.

```
qrun -H
```

PBS runs the job regardless of scheduling policy and available resources.
Chapter 4  Scheduling

The `qrun` command alone overrides the following:

- Limits on resource usage by users, groups, and projects
- Limits on the number of jobs that can be run at a vnode
- Boundaries between primetime and non-primetime, specified in `backfill_prime`
- Whether the job is in a primetime queue: you can run a job in a primetime queue even when it’s not primetime, or vice versa. Primetime boundaries are not honored.
- Dedicated time: you can run a job in a dedicated time queue, even if it’s not in a dedicated time queue, and vice versa. However, dedicated time boundaries are still honored.

The `qrun` command alone does not override the following:

- Server and queue resource usage limits

4.8.30.1.i  Using `qrun` Without `-H` Option on Shrink-to-fit Jobs

When a shrink-to-fit job is run via `qrun`, and there is a hard deadline, e.g. reservation or dedicated time, that conflicts with the shrink-to-fit job’s `max_walltime` but not its `min_walltime`, the following happens:

- If preemption is enabled and there is a preemptable job before the hard deadline that must be preempted in order to run the shrink-to-fit job, preemption behavior means that the shrink-to-fit job does not shrink to fit; instead, it conflicts with the deadline and does not run.
- If there is no preemptable job before the hard deadline, the shrink-to-fit job shrinks into the available time and runs.

4.8.30.1.ii  Using `qrun` With `-H` Option on Shrink-to-fit Jobs

When a shrink-to-fit job is run via `qrun -H`, the shrink-to-fit job runs, regardless of reservations, dedicated time, other jobs, etc. When run via `qrun -H`, shrink-to-fit jobs do not shrink. If the shrink-to-fit job has a requested or inherited value for `walltime`, that value is used, instead of one set by PBS when the job runs. If no `walltime` is specified, the job runs without a `walltime`.


4.8.30.1.iii  `qrun` Caveats

- A job that has just been run via `qrun` has top priority only during the scheduling cycle where it was `qrun`. At the next scheduling cycle, that job is available for preemption just like any other job.
- Be careful when using `qrun -H` on jobs or vnodes involved in reservations.
4.8.30.2  Hold a Job Manually

You can use the `qhold` command to place a hold on a job. The effect of placing a hold depends on whether the job is running and whether you have checkpointing configured:

- If the job is queued, the job will not run.
- If the job is running and checkpoint-abort is configured, the job is checkpointed, requeued, and held.
- If the job is running and checkpoint-abort is not configured, the only change is that the job's `Hold_Types` attribute is set to User Hold. If the job is subsequently requeued, it will not run until the hold is released.

You can release the hold using the `qrls` command.

For information on checkpointing jobs, see section 9.3, “Checkpoint and Restart”, on page 711.


4.8.30.3  Suspend a Job Manually

You can use the `qsig -s suspend` command to suspend a job so that it won’t run. If you suspend a job, and then release it using the `qsig -s resume` command, the job remains in the suspended state until the required resources are available.

You can resume the job immediately by doing the following:

1. Resume the job:
   ```bash
   qsig -s resume <job ID>
   ```
2. Run the job manually:
   ```bash
   qrun <job ID>
   ```

See “qsig” on page 192 of the PBS Professional Reference Guide.
4.8.30.4 Set Special Resource Value Used in Formula

You can change the value of a resource used in the job sorting formula. For example, to give a particular job a higher priority by changing the value of a custom resource called “higher”:

- Create a custom resource that is invisible to job submitters in `resourcedef`:
  ```
  higher type=float flag=i
  ```
- Restart the server. See section 5.14.3.1, “Restarting the Server”, on page 332.
- The formula expression includes “higher”:
  ```
  Qmgr: s s job_sort_formula = “higher”
  ```
- Set the default for this resource at the server:
  ```
  Qmgr: set server resources_default.higher = 1
  ```
- These jobs are submitted:
  ```
  Job 1
  Job 2
  Job 3
  ```
- Change Job 2 so that its value for “higher” is 5:
  ```
  qalter -l higher = 5 job2
  ```
- The scheduler logs the following:
  ```
  Job ;1.host1;Formula Evaluation = 1
  Job ;2.host1;Formula Evaluation = 5
  Job; 3.host1;Formula Evaluation = 1
  ```
- Jobs are sorted in this order:
  ```
  Job 2
  Job 1
  Job 3
  ```

4.8.30.5 Change Formula On the Fly

You can change the job sorting formula on the fly, so that the next scheduler iteration uses your new formula. This will change how job priorities are computed, and can rearrange the order in which jobs are run. See section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.
4.8.30.6 Using Dedicated Time

You can set up blocks of dedicated time, where the only jobs eligible to be started or running are the ones in dedicated time queues. You can use dedicated time for upgrades. See section 4.8.10, “Dedicated Time”, on page 153, and section 2.2.5.2.i, “Dedicated Time Queues”, on page 23.

4.8.30.7 Using cron Jobs or the Windows Task Scheduler

You can use cron jobs or the Windows Task Scheduler to change PBS settings according to the needs of your time slots. See section 4.8.7, “cron Jobs, or the Windows Task Scheduler”, on page 132.

4.8.30.8 Using Hooks

You can use hooks to examine jobs and alter their characteristics. See Chapter 6, "Hooks", on page 413.

4.8.31 Peer Scheduling

Peer scheduling allows separate PBS complexes to automatically run jobs from each other’s queues. This means that you can dynamically balance the workload across multiple, separate PBS complexes. These cooperating PBS complexes are referred to as “Peers”.

4.8.31.1 How Peer Scheduling Works

In peer scheduling, a PBS server pulls jobs from one or more peer servers and runs them locally. When Complex A pulls a job from Complex B, Complex A is the “pulling” complex and Complex B is the “furnishing” complex. When the pulling scheduler determines that another complex’s job can immediately run locally, it moves the job to the specified queue on the pulling server and immediately run the job. The job is run as if it had been submitted to the pulling complex.

You can set up peer scheduling so that A pulls from B and C, and so that B also pulls from A and C.

A job is pulled only when it can run immediately.

The pulling complex must have all of the resources required by the job, including custom resources.
When a job is pulled from one complex to another, the pulling complex applies its policy to the job. The job's execution priority is determined by the policy of the pulling complex. You can set special priority for pulled jobs; see section 4.8.31.4.ii, “Setting Priority for Pulled Jobs”, on page 209.

4.8.31.2 Prerequisites for Peer Scheduling

- You must create the pulling and furnishing queues before peer scheduling can be configured. See section 2.2.3, “Creating Queues”, on page 21 on how to create queues.
- When configuring peer scheduling, it is strongly recommended to use the same version of PBS Professional at all peer locations.
- Make sure that custom resources are consistent across peer locations. Jobs requesting custom resources at one location will not be able to run at another unless the same resources are available.
- Under Windows, if single_signon_password_enable is set to True among all peer servers, users must have their password cached on each server. See section 8.11.1.1, “Per-user/per-server Passwords”, on page 681.

4.8.31.3 Configuring Peer Scheduling

The following sections give details on how to configure peer scheduling. Here is a brief outline:

- Define a flat user namespace on all complexes
- Map pulling queues to furnishing queues
- If necessary, specify port
- Grant manager access to each pulling server
- If possible, make user-to-group mappings be consistent across complexes
- If any of the peering sites is using failover, configure peering to work with failover

4.8.31.3.i Defining a Flat User Namespace

Peer scheduling requires a flat user namespace in all complexes involved. This means that user “joe” on the remote peer system(s) must be the same as user “joe” on the local system. Your site must have the same mapping of user to UID across all hosts, and a one-to-one mapping of UIDs to usernames. It means that PBS does not need to check whether X@hostA is the same as X@hostB; it can just assume that this is true. Set flatuid to True:

`Qmgr: set server flatuid = True`

For more on flatuid, see section 8.3.13, “Flatuid and Access”, on page 667.
Mapping Pulling Queues to Furnishing Queues

You configure peer scheduling by mapping a furnishing peer’s queue to a pulling peer’s queue. You can map each pulling queue to more than one furnishing queue, or more than one pulling queue to each furnishing queue.

The pulling and furnishing queues must be execution queues, not route queues. However, the queues can be either ordinary queues that the complex uses for normal work, or special queues set up just for peer scheduling.

You map pulling queues to furnishing queues by setting the `peer_queue` scheduler configuration option in `PBS_HOME/sched_priv/sched_config`. The format is:

```
peer_queue: "<pulling queue> <furnishing queue>@<furnishing server>.domain"
```

For example, Complex A’s queue “workq” is to pull from two queues: Complex B’s queue “workq” and Complex C’s queue “slowq”. Complex B’s server is ServerB and Complex C’s server is ServerC. You would add this to Complex A’s `PBS_HOME/sched_priv/sched_config`:

```
peer_queue: "workq workq@ServerB.domain.com"
peer_queue: "workq slowq@ServerC.domain.com"
```

Or if you wish to direct Complex B’s jobs to queue Q1 on Complex A, and Complex C’s jobs to Q2 on Complex A:

```
peer_queue: "Q1 workq@ServerB.domain.com"
peer_queue: "Q2 fastq@ServerC.domain.com"
```

In one complex, you can create up to 50 mappings between queues. This means that you can have up to 50 lines in `PBS_HOME/sched_priv/sched_config` beginning with `peer_queue`.

Specifying Ports

The default port for the server to listen on is 15001, and the scheduler uses any privileged port (1023 and lower). If the furnishing server is not using the default port, you must specify the port when you specify the queue. For example, if ServerB is using port 16001, and you wish to pull jobs from workq at ServerB to workq at ServerA, add this to `PBS_HOME/sched_priv/sched_config` at ServerA:

```
peer_queue: "workq workq@ServerB.domain.com:16001"
```

The scheduler and server communicate via TCP.
4.8.31.3.iv  Granting Manager Access to Pulling Servers

Each furnishing server must grant manager access to each pulling server. If you wish jobs to move in both directions, where Complex A will both pull from and furnish jobs to Complex B, ServerA and ServerB must grant manager access to each other.

On the furnishing complex:

For UNIX:

```
Qmgr: set server managers += root@pullingServer.domain.com
```

For Windows:

```
Qmgr: set server managers += <name of PBS service account>@*
```

4.8.31.3.v  Making User-to-group Mappings Consistent Across Complexes

If possible, ensure that for each user in a peer complex, that user is in the same group in all participating complexes. So if user “joe” is in groupX on Complex A, user “joe” should be in groupX on Complex B. This means that a job’s egrou p attribute will be the same on both complexes, and any group limit enforcement can be properly applied.

There is a condition when using peer scheduling in which group hard limits may not be applied correctly. This can occur when a job’s effective group, which is its egrou p attribute, i.e. the job’s owner’s group, is different on the furnishing and pulling systems. When the job is moved over to the pulling complex, it can evade group limit enforcement if the group under which it will run on the pulling system has not reached its hard limit. The reverse is also true; if the group under which it will run on the pulling system has already reached its hard limit, the job won’t be pulled to run, although it should.

This situation can also occur if the user explicitly specifies a group via qsub -W group_list.

It is recommended to advise users to not use the qsub options “-u user_list” or “-W group_list=groups” in conjunction with peer scheduling.

4.8.31.3.vi  Configuring Peer Scheduling with Failover

If you are configuring peer scheduling so that Complex A will pull from Complex B where Complex B is configured for failover, you must configure Complex A to pull from both of Complex B’s servers. For these instructions, see section 9.2.6.2, “Configuring Failover to Work With Peer Scheduling”, on page 708.
4.8.31.4 Peer Scheduling Advice

4.8.31.4.i Selective Peer Scheduling

You can choose the kinds of jobs that can be selected for peer scheduling to a different complex. You can do the following:

• Set resource limits at the furnishing queue via the `resources_min` and `resources_max` queue attributes. See section 2.2.6.4, “Using Resources to Route Jobs Between Queues”, on page 25.
• Route jobs into the furnishing queue via a hook. See section 6.7.1, “Routing Jobs Using Hooks”, on page 426.
• Route jobs into the furnishing queue via a routing queue. See section 2.2.6, “Routing Queues”, on page 24.

4.8.31.4.ii Setting Priority for Pulled Jobs

You can set a special priority for pulled jobs by creating a queue that is used only as a pulling queue, and setting the pulling queue’s priority to the desired level. You can then use the queue’s priority when setting job execution priority. See section 4.2.5.3.iv, “Using Queue Priority when Computing Job Priority”, on page 69.

For example, if you give the pulling queue the lowest priority, the pulling complex will pull a job only when there are no higher-priority jobs that can run.

You can also have pulled jobs land in a special queue where they inherit a custom resource that is used in the job sorting formula.

4.8.31.5 How Peer Scheduling Affects Jobs

4.8.31.5.i How Peer Scheduling Affects Inherited Resources

If the job is moved from one server to another via peer scheduling, any default resources in the job’s resource list inherited from the furnishing queue or server are removed. This includes any select specification and place directive that may have been generated by the rules for conversion from the old syntax. If a job's resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either `select` or `place` is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.

When the pulling scheduler runs the job the first time, the job is run as if the job still had all of the resources it had at the furnishing complex. If the job is requeued and restarted at the pulling complex, the job picks up new default resources from the pulling complex, and is scheduled according to the newly-inherited resources from the pulling complex.
4.8.31.5.ii How Peer Scheduling Affects Policy Applied to Job

After a job is pulled from one complex to another, the scheduling policy of the pulling complex is applied to the job.

For example, if you use queue priority in the formula and the job is moved to another server through peer scheduling, the queue priority used in the formula will be that of the queue to which the job is moved.

When a job is pulled from one complex to another, hooks are applied at the new complex as if the job had been submitted locally. For example, if the pulling complex has a queuejob hook, that hook runs when a job is pulled.

4.8.31.5.iii How Peer Scheduling Affects Job’s Eligible Time

The job’s eligible_time is preserved when a job is moved due to peer scheduling.

4.8.31.5.iv Viewing Jobs That Have Been Moved to Another Server

If you are connected to ServerA and a job submitted to ServerA has been moved from ServerA to ServerB through peer scheduling, in order to display it via qstat, give the job ID as an argument to qstat. If you only give the qstat command, the job will not appear to exist. For example, the job 123.ServerA is moved to ServerB. In this case, use

```
qstat 123
```

or

```
qstat 123.ServerA
```

To list all jobs at ServerB, you can use:

```
qstat @ServerB
```

4.8.31.5.v Peer Scheduling and Hooks

When a job is pulled from one complex to another, the following happens:

- Hooks are applied at the new complex as if the job had been submitted locally
- Any movejob hooks at the furnishing server are run

4.8.31.6 Peer Scheduling Caveats

- Each complex can peer with at most 50 other complexes.
- When using peer scheduling, group hard limits may not be applied correctly. This can occur when the job owner’s group is different on the furnishing and pulling systems. For help in avoiding this problem, see section 4.8.31.3.v, “Making User-to-group Mappings.”
You cannot peer schedule between a Windows complex and a UNIX/Linux complex.

When the pulling scheduler runs the job the first time, the job is run as if the job still had all of the resources it had at the furnishing complex. If the job is requeued and restarted at the pulling complex, the job picks up new default resources from the pulling complex, and is scheduled according to the newly-inherited resources from the pulling complex.

Peer scheduling is not supported for job arrays.

### 4.8.32 Placement Sets

Placement sets are the sets of vnodes within which PBS will try to place a job. PBS tries to group vnodes into the most useful sets, according to how well connected the vnodes are, or the values of resources available at the vnodes. Placement sets are used to improve task placement (optimizing to provide a “good fit”) by exposing information on system configuration and topology. The scheduler tries to put a job in the smallest appropriate placement set.

#### 4.8.32.1 Definitions

**Task placement**

The process of choosing a set of vnodes to allocate to a job that will satisfy both the job’s resource request (select and place specifications) and the configured scheduling policy.

**Placement Set**

A set of vnodes. Placement sets are defined by the values of vnode-level string array resources. A placement set is all of the vnodes that have the same value for a specified defining resource substring. For example, if the defining resource is a vnode-level string array named “switch”, which can have values “S1”, “S2”, or “S3”: the set of vnodes which have a substring matching “switch=S2” is a placement set.

Placement sets can be specified at the server or queue level.

**Placement Set Series**

A set of sets of vnodes.

A placement set series is all of the placement sets that are defined by specifying one string array resource. Each placement set in the series is the set of vnodes that share one value for the resource. There is one placement set for each value of the resource. If the resource takes on N values at the vnodes, then there are N sets in the series. For example, if the defining resource is a string array named “switch”, which can have values “S1”, “S2”, or “S3”, there are three sets in the series. The first is defined by the value “S1”, where all the vnodes in that set...
have the value “S1” for the resource switch. The second set is defined by “S2”, and the third by “S3”.

Each of the resources named in node_group_key specifies a placement series. For example, if the server’s node_group_key attribute contains “router,switch”, then the server has two placement set series.

Placement Pool

All of the placement sets that are defined; the server can have a placement pool, and each queue can have its own placement pool. If a queue has no placement pool, the scheduler uses the server’s placement pool.

A placement pool is the set of placement set series that are defined by one or more string array resources named in node_group_key.

For example, if the server’s node_group_key attribute contains “router,switch”, and router can take the values “R1” and “R2” and switch can take the values “S1”, “S2”, and “S3”, then there are five placement sets, in two placement series, in the server’s placement pool.

Static Fit

A job statically fits into a placement set if the job could fit into the placement set if the set were empty. It might not fit right now with the currently available resources.

Dynamic Fit

A job dynamically fits into a placement set if it will fit with the currently available resources (i.e. the job can fit right now).

4.8.32.2 Requirements for Placement Sets

- Placement sets are enabled by setting the server’s node_group_enable attribute to True
- Server-level placement sets are defined by setting the server’s node_group_key attribute to a list of vnode-level string array resources.
- Queue-level placement sets are defined by setting a queue’s node_group_key attribute to a list of vnode-level string array resources.
- At least one vnode-level string array resource must exist on vnodes and be set to values that can be used to partition the vnodes.
4.8.32.3 Description of Placement Sets

4.8.32.3.i What Defines a Placement Set, Series, or Pool

Placement sets are defined by the values of vnode-level string array resources. You define placement sets by specifying the names of these resources in the node_group_key attribute for the server and/or queues. Each value of each resource defines a different placement set. A placement set is all of the vnodes that have the same value for a specified defining resource. For example, if the defining resource is a vnode-level string array named “switch”, which has the values “S1”, “S2”, and “S3”, the set of vnodes where switch has the value “S2” is a placement set. If some vnodes have more than one substring, and one of those substrings is the same in each vnode, those vnodes make up a placement set. For example, if the resource is “router”, and vnode V0 has resources_available.router set to “r1i0,r1”, and vnode V1 has resources_available.router set to “r1i1,r1”, V0 and V1 are in the placement set defined by resources_available.router = “r1”. If the resource has N distinct values across the vnodes, including the value zero and being unset, there are N placement sets defined by that resource.

Each placement set can have a different number of vnodes; the number of vnodes is determined only by how many vnodes share that resource value.

Each placement set series is defined by the values of a single resource across all the vnodes. For example, if there are three switches, S1, S2 and S3, and there are vnodes with resources_available.switch that take on one or more of these three values, then there will be three placement sets in the series.

Whenever you define any placement sets, you are defining a placement pool. Placement pools can be defined for the server and for each queue. You define a server-level placement pool by setting the server’s node_group_key to a list of one or more vnode-level string array resources. You define a queue-level placement pool by similarly setting the queue’s node_group_key.

4.8.32.3.ii Vnode Participation in Placement Sets, Series, and Pools

Each vnode can be in multiple placement sets, placement set series, and placement pools.

A vnode can be in multiple placement sets in the same placement set series. For example, if the resource is called “router”, and a vnode’s router resource is set to “R1, R2”, then the vnode will be in the placement set defined by router = R1 and the set defined by router = R2.

A vnode is in a placement series whenever the resource that defines the series is defined on the vnode. For example, if placement sets are defined by the values of the router and the switch resources, and a vnode has value R1 for router, and S1 for switch, then the vnode is in both placement series, because it is in the set that shares the R1 value for router, and the set that shares the S1 value for switch. Each of those sets is one of a different series.
Chapter 4  Scheduling

The server has its own placement pool if the server’s node_group_key attribute is set to at least one vnode-level string array resource. Similarly, each queue can have its own placement pool. A vnode can be in any placement pool that specifies a resource that is defined on the vnode.

4.8.32.3.iii  Multihost Placement Sets

Placement sets, series, and pools can span hosts. Placement sets can be made up of vnodes from anywhere, regardless of whether the vnode is from a multi-vnode host.

To set up a multihost placement set, choose a string array resource for the purpose, and list it in the desired node_group_key attribute. For example, create a string_array resource called “span” in the PBS_HOME/server_priv/resourcedef file:

```
span type=string_array flag=h
```

Restart the server. See section 5.14.3.1, “Restarting the Server”, on page 332. Add the resource “span” to node_group_key on the server or queue. Use qmgr to give it the same value on all the desired vnodes. You can write a script that sets the same value on each vnode that you want in your placement set.

4.8.32.3.iv  Machines with Multiple Vnodes

Machines with multiple vnodes such as the SGI Altix are represented as a generic set of vnodes. Placement sets are used to allocate resources on a single machine to improve performance and satisfy scheduling policy and other constraints. Jobs are placed on vnodes using placement set information. For placement set generation information for SGI machines, see section 10.4.8.1, “Generation of Placement Set Information”, on page 812.

4.8.32.3.v  Placement Sets Defined by Unset Resources

For each defining resource, vnodes where that resource is unset are grouped into their own placement set. For example, if you have ten vnodes, on which there is a string resource COLOR, where two have COLOR set to “red”, two are set to “blue”, two are set to “green” and the rest are unset, there will be four placement sets defined by the resource COLOR. This is because the fourth placement set consists of the four vnodes where COLOR is unset. This placement set will also be the largest.

Every resource listed in node_group_key could potentially define such a placement set.

4.8.32.3.vi  Placement Sets and Node Grouping

Node grouping is the same as one placement set series, where the placement sets are defined by a single resource. Node grouping has been superseded by placement sets.

In order to have the same behavior as in the old node grouping, group on a single resource. If this resource is a string array, it should only have one value on each vnode. This way, each vnode will only be in one node group.
### 4.8.32.4 How Placement Sets Are Used

You use placement sets to partition vnodes according to the value of one or more resources. Placement sets allow you to group vnodes into useful sets.

You can run multi-vnode jobs in one placement set. For example, it makes the most sense to run a multi-vnode job on vnodes that are all connected to the same high-speed switch.

PBS will attempt to place each job in the smallest possible set that is appropriate for the job.

#### 4.8.32.4.i Order of Placement Pool Selection

The scheduler chooses one placement pool from which to select a placement set. If the job cannot run in that placement pool, the scheduler ignores placement sets for the job.

Queue placement pools override the server’s placement pool. If a queue has a placement pool, jobs from that queue are placed using the queue’s placement pool. If a queue has no placement pool (the queue’s node_group_key is not defined), jobs are placed using the server’s placement pool, if it exists.

A per-job placement set is defined by the -l place statement in the job’s resource request. Since the job can only request one value for the resource, it can only request one specific placement set. A job’s place=group resource request overrides the sets defined by the queue’s or server’s node_group_key.

The scheduler chooses the most specific placement pool available, following this order of precedence:

1. A per-job placement set (job’s place=group= request)
2. A placement set from the placement pool for the job’s queue
3. A placement set from the complex-wide placement pool
4. All vnodes in the complex

#### 4.8.32.4.ii Order of Placement Set Consideration Within Pool

The scheduler looks in the selected placement pool and chooses the smallest possible placement set that is appropriate for the job. The scheduler examines the placement sets in the pool and orders them, from smallest to largest, according to the following rules:

1. Static total ncpus of all vnodes in set
2. Static total mem of all vnodes in set
3. Dynamic free ncpus of all vnodes in set
4. Dynamic free mem of all vnodes in set
If a job can fit statically within any of the placement sets in the placement pool, then the scheduler places a job in the first placement set in which it fits dynamically. This ordering ensures the scheduler will use the smallest possible placement set in which the job will dynamically fit. If there are multiple placement sets where the job fits statically, but some are being used, the scheduler uses the first placement set where the job can run now. If the job fits statically into at least one placement set, but these placement sets are all busy, the scheduler waits until a placement set can fit the job dynamically.

If a job cannot statically fit into any placement set in the selected placement pool, the scheduler ignores defined placement sets and uses all available vnodes as its placement set, unless the do_not_span_psets scheduler attribute is True, in which case the job will not run.

For example, we have the following placement sets, and a job that requests 8 CPUs:

Set1 ncpus = 4
Set2 ncpus = 12; this placement set is full
Set3 ncpus = 16; this placement set is not being used

The scheduler looks at Set1; Set1 is statically too small, and the scheduler moves to the next placement set. Set2 is statically large enough, but the job does not fit dynamically. The scheduler looks at Set3; Set3 is large enough, and the job fits dynamically. The scheduler runs the job in Set3.

If the job requests 24 CPUs, the scheduler attempts to run the job in the set consisting of all vnodes.

4.8.32.4.iii Order of Vnode Selection Within Set

The scheduler orders the vnodes within the selected placement set using the following rules:

- If node_sort_key is set, vnodes are sorted by node_sort_key. See section 4.8.48, “Sorting Vnodes on a Key”, on page 283.
- If node_sort_key is not set, the order in which the vnodes are returned by pbs_statnode(). This is the default order the vnodes appear in the output of the pbsnodes -a command.

The scheduler places the job on the vnodes according to their ordering above.

4.8.32.5 Summary of Placement Set Requirements

The steps to configure placement sets are given in the next section. The requirements are summarized here for convenience:

- A PBS_HOME/server_priv/resourcedef file containing definitions of the
resources of interest
- You must restart the server after new definition

- Vnodes defining a value for each resource to be used for placement sets (e.g., rack)
  - If defined via vnode definition, you must HUP the MoMs involved

- The server’s or queue’s node_group_key attribute must be set to the resources to be used for placement sets. For example, if we have custom resources named “rack”, “socket”, “board”, and “boardpair”, which are to be used for placement sets:

  ```bash
  Qmgr: set server node_group_key = "rack,socket,board,boardpair"
  ```
  - No signals needed, takes effect immediately

- Placement sets must be enabled at the server by setting the server’s node_group_enable attribute to True. For example:

  ```bash
  Qmgr: set server node_group_enable=True
  ```
  - No signals needed, takes effect immediately

Adding a resource to the scheduler’s resources line is required only if the resource is to be specifically requested by jobs. It is not required for -1place=group=<resource>. 
4.8.32.6 How to Configure Placement Sets

The following steps show how to satisfy the requirements for placement sets:

1. If the vnodes that you will use in placement sets are not defined, define them. See section 3.1.5, “Creating Vnodes”, on page 39.

2. If the vnode-level string array resources that you will use to define placement sets do not exist, create them. See section 5.14.5, “Configuring Host-level Custom Resources”, on page 336.

3. Restart the server; see section 5.14.3.1, “Restarting the Server”, on page 332.

4. If values for the vnode-level string array resources that you will use to define placement sets are not set at the vnodes you wish to use, set the values. See section 3.5, “How to Configure MoMs and Vnodes”, on page 50.

5. If you use vnode definition files to set values for vnode-level string array resources, HUP the MoMs involved.

6. To create queue placement pools, set the node_group_key attribute to the name(s) of one or more vnode-level string array resources. Do this for each queue for which you want a separate pool. For example:

```
Qmgr: set queue workq node_group_key = <router,switch>
```

7. To create a server placement pool, set the node_group_key server attribute to the name(s) of one or more vnode-level string array resources. For example:

```
Qmgr: set server node_group_key = <router,switch>
```

For example, to create a server-level placement pool for the resources host, L2 and L3:

```
Qmgr: set server node_group_key = "host,L2,L3"
```

8. Set the server’s node_group_enable attribute to True

```
Qmgr: set server node_group_enable = True
```

9. For ease of reviewing placement set information, you can add the name of each resource used in each vnode’s pnames attribute:

```
Qmgr: active node <vnode name>,<vnode name>,...
Qmgr: set node pnames += <resource name>
```

or

```
Qmgr: set node pnames = <resource list>
```

For example:

```
Qmgr: set node pnames =
We recommend using the natural vnode for any placement set information that is invariant for a given host.

Resources used only for defining placement sets, and not for allocation to jobs, do not need to be listed in the resources: line in PBS_HOME/sched_priv/sched_config. So for example if you create a resource just for defining placement sets, and jobs will not be requesting this resource, you do not need to list it in the resources: line.
4.8.32.7 Examples of Creating Placement Sets

4.8.32.7.i Cluster with Four Switches

This cluster is arranged as shown with vnodes 1-4 on Switch1, vnodes 5-10 on Switch2, and vnodes 11-24 on Switch3. Switch1 and Switch2 are on Switch4.

To make the placement sets group the vnodes as they are grouped on the switches:

Create a custom resource called `switch`. The `-h` flag makes the resource requestable:

```bash
switch type=string_array flag=h
```
On vnodes[1-4] set:

    resources_available.switch="switch1,switch4"

On vnodes[5-10] set:

    resources_available.switch="switch2,switch4"

On vnodes[11-24] set:

    resources_available.switch="switch3"

On the server set:

    node_group_enable=True
    node_group_key=switch

So you have 4 placement sets:

- The placement set "switch1" has 4 vnodes
- The placement set "switch2" has 6 vnodes
- The placement set "switch3" has 14 vnodes
- The placement set "switch4" has 10 vnodes

PBS will try to place a job in the smallest available placement set. Does the job fit into the smallest set (switch1)? If not, does it fit into the next smallest set (switch2)? This continues until it finds one where the job will fit.

4.8.32.7.ii Example of Configuring Placement Sets on an SGI Altix

For information on how to configure vnodes on a cpusetted machine in order to define new placement sets on an Altix, use the instructions in section 3.5.2.3, “Configuring Machines with Cpusets”, on page 52.
Scheduling

Chapter 4

In this example, we define a new placement set using the new resource “NewRes”. We create
a file called SetDefs that contains the changes we want.
1.

Add the new resource to the server’s resourcedef file:
NewRes type=string_array flag=h

2.

Restart the server. See section 5.14.3.1, “Restarting the Server”, on page 332.

3.

Add NewRes to the server's node_group_key
Qmgr: set server node_group_key+="NewRes"

4.

Add NewRes to the value of the pnames attribute for the natural vnode. This makes the
name of the resource you used easily available. Add a line like this to SetDefs:
altix3: resources_available.pnames =”...,NewRes”

5.

For each vnode, V, that's a member of a new placement set you're defining, add a line of
the form:
V: resources_available.NewRes = “<value1[,...]>”

All the vnodes in the new set should have lines of that form, with the same resource
value, in the new configuration file.
Here the value of the resource is “P” and/or “Q”.
We’ll put vnodes A, B and C into one placement set, and vnodes B, C and D into another.
A:
B:
C:
D:

resources_available.NewRes2
resources_available.NewRes2
resources_available.NewRes2
resources_available.NewRes2

=
=
=
=

P
“P,Q”
“P,Q”
Q

For each new placement set you define, use a different value for the resource.
6.

Add SetDefs and tell MoM to read it, to make a Version 2 MoM configuration file
NewConfig.
pbs_mom -s insert NewConfig SetDefs

7.

Stop and restart the MoM. See “Starting and Stopping PBS: UNIX and Linux” on page
177 in the PBS Professional Installation & Upgrade Guide.

4.8.32.7.iii

Example of Altix Placement Pool

In this example, we have vnodes connected to four cbricks and two L2 connectors. Since
these come from the MoM, they are automatically added to the server’s resourcedef file.
Enable placement sets:
Qmgr: s s node_group_enable=True

222

PBS Professional 12.2 Administrator’s Guide


Define the pool you want:

```
Qmgr: s s node_group_key="cbrick, L2"
```

When you use the following:

```
pbsnodes -av | egrep '([^ ]|) | cbrick
```

or

```
pbsnodes -av | egrep '([^ ]|) | L2
```

and the vnodes look like this:

- vnode1
  - resources_available.cbrick=cbrick1
  - resources_available.L2=A

- vnode2
  - resources_available.cbrick=cbrick1
  - resources_available.L2=B

- vnode3
  - resources_available.cbrick=cbrick2
  - resources_available.L2=A

- vnode4
  - resources_available.cbrick=cbrick2
  - resources_available.L2=B

- vnode5
  - resources_available.cbrick=cbrick3
  - resources_available.L2=A

- vnode6
  - resources_available.cbrick=cbrick3
  - resources_available.L2=B

- vnode7
  - resources_available.cbrick=cbrick4
  - resources_available.L2=A

- vnode8
  - resources_available.cbrick=cbrick4
  - resources_available.L2=B
There are six resulting placement sets.

\begin{itemize}
  \item \texttt{cbrick=cbrick1: \{vnode1, vnode2\}}
  \item \texttt{cbrick=cbrick2: \{vnode3, vnode4\}}
  \item \texttt{cbrick=cbrick3: \{vnode5, vnode6\}}
  \item \texttt{cbrick=cbrick4: \{vnode7, vnode8\}}
  \item \texttt{L2=A: \{vnode1, vnode3, vnode5, vnode7\}}
  \item \texttt{L2=B: \{vnode2, vnode4, vnode6, vnode8\}}
\end{itemize}

### Example of Placement Sets Using Colors

A placement pool is defined by two resources: \texttt{colorset1} and \texttt{colorset2}, by using "\texttt{node_group_key=colorset1,colorset2}".  

If a vnode has the following values set:

\begin{itemize}
  \item \texttt{resources_available.colorset1=blue, red}
  \item \texttt{resources_available.colorset2=green}
\end{itemize}

The placement pool contains at least three placement sets. These are:

\begin{itemize}
  \item \texttt{\{resources_available.colorset1=blue\}}
  \item \texttt{\{resources_available.colorset1=red\}}
  \item \texttt{\{resources_available.colorset2=green\}}
\end{itemize}

This means the vnode is in all three placement sets. The same result would be given by using one resource and setting it to all three values, e.g. \texttt{colorset=blue,red,green}.

Example: We have five vnodes \texttt{v1 – v5}:

\begin{itemize}
  \item \texttt{v1 color=red host=mars}
  \item \texttt{v2 color=red host=mars}
  \item \texttt{v3 color=red host=venus}
  \item \texttt{v4 color=blue host=mars}
  \item \texttt{v5 color=blue host=mars}
\end{itemize}

The placement sets are defined by

\begin{itemize}
  \item \texttt{node_group_key=color}
\end{itemize}

The resulting node groups would be: \texttt{\{v1, v2, v3\}}, \texttt{\{v4, v5\}}

### Simple Switch Placement Set Example

Say you have a cluster with two high-performance switches each with half the vnodes connected to it. Now you want to set up placement sets so that jobs will be scheduled only onto the same switch.

Next, we need to enable placement sets and specify the resource to use:

```
Qmgr: set server node_group_enable=True
Qmgr: set server node_group_key=switch
```

Now, set the value for switch on each vnode:

```
Qmgr: active node vnode1,vnode2,vnode3
Qmgr: set node resources_available.switch=A
Qmgr: active node vnode4,vnode5,vnode6
Qmgr: set node resources_available.switch=B
```

Now there are two placement sets:

- switch=A: {vnode1, vnode2, vnode3}
- switch=B: {vnode4, vnode5, vnode6}

### 4.8.32.8 Placement Sets and Reservations

When PBS chooses a placement set for a reservation, it makes the same choices as it would for a regular job. It fits the reservation into the smallest possible placement set. See section 4.8.32.4.ii, “Order of Placement Set Consideration Within Pool”, on page 215.

When a reservation is created, it is created within a placement set, if possible. If no placement set will satisfy the reservation, placement sets are ignored. The vnodes allocated to a reservation are used as one single placement set for jobs in the reservation; they are not subdivided into smaller placement sets. A job within a reservation runs within the single placement set made up of the vnodes allocated to the reservation.

### 4.8.32.9 Placement Sets and Load Balancing

If you configure both placement sets and load balancing, the net effect is that vnodes that are over their load limit will be removed from consideration.

### 4.8.32.10 Viewing Placement Set Information

You can find information about placement sets in the following places:

- The server’s `node_group_enable` attribute shows whether placement sets are enabled
- The server’s `node_group_key` attribute contains the names of resources used for that queue’s placement pool
- Each queue’s `node_group_key` attribute contains the names of resources used for that
Each vnode's `pnames` attribute can contain the names of resources used for placement sets, if properly set.

- PBS-generated MoM configuration files contain names and values of resources.

### 4.8.32.11 Placement Set Caveats and Advice

- When you create a Version 2 configuration file for a pre-existing vnode, make sure it specifies all of the information about the vnode, such as resources and attribute settings. The creation of the configuration file overrides previous settings, and if the new file contains no specification for a resource or attribute, that resource or attribute becomes unset.

- If there is a vnode-level platform-specific resource set on the vnodes on the Altix, then `node_group_key` should probably include this resource, because this will enable PBS to run jobs in more logical sets of vnodes.

- If the user specifies a job-specific placement set, for example `place=group=switch`, but the job cannot statically fit into any switch placement set, then the job will still run, but not in a switch placement set.

- The `pnames` vnode attribute is for displaying to the administrator the resources used for placement sets. This attribute is not used by PBS.

### 4.8.32.11.i Non-backward-compatible Change in Node Grouping

Given the following example configuration:

- vnode1: switch=A
- vnode2: switch=A
- vnode3: switch=B
- vnode4: switch=B
- vnode5: switch unset

Qmgr: `s s node_group_key=switch`

There is no change in the behavior of jobs submitted with `qsub -l ncpus=1`:

- version 7.1: The job can run on any node: node1, ..., node5
- version 8.0: The job can run on any node: node1, ..., node5

Example of 8.0 and later behavior: jobs submitted with `qsub -l nodes=1`:

- version 7.1: The job can only run on nodes: node1, node2, node3, node4. It will never use node5
- version 8.0: The job can run on any node: node1, ..., node5
Overall, the change for version 8.0 was to include every vnode in placement sets (when enabled). In particular, if a resource is used in node_group_key, PBS will treat every vnode as having a value for that resource, hence every vnode will appear in at least one placement set for every resource. For vnodes where a string resource is "unset", PBS will behave as if the value is "".

4.8.32.12 Attributes and Parameters Affecting Placement Sets

**do_not_span_psets**
Scheduler attribute. Specifies whether or not the scheduler requires the job to fit within one of the existing placement sets. When do_not_span_psets is set to True, the scheduler will require the job to fit within a single existing placement set. The scheduler checks all placement sets, whether or not they are currently in use. If the job fits in a currently-used placement set, the job must wait for the placement set to be available. If the job cannot fit within a single placement set, it will not run.

When this attribute is set to False, the scheduler will first attempt to place the job in a single placement set, but if it cannot, it will allow the job to span placement sets, running on whichever vnodes can satisfy the job’s resource request.

Format: Boolean
Default value: False (This matches behavior of PBS 10.4 and earlier)

Example: To require jobs to fit within one placement set:

```bash
Qmgr: set sched do_not_span_psets=True
```

**node_group_enable**
Server attribute. Specifies whether placement sets are enabled.

Format: Boolean
Default: False

**node_group_key**
Server and queues have this attribute. Specifies resources to use for placement set definition. Queue’s attribute overrides server’s attribute.

Format: string_array
Default: Unset
4.8.32.13 Errors and Logging

If do_not_span_psets is set to True, and a job requests more resources than are available in one placement set, the following happens:

- The job's comment is set to the following:
  "Not Running: can't fit in the largest placement set, and can't span psets"
- The following message is printed to the scheduler's log:
  "Can't fit in the largest placement set, and can't span placement sets"

4.8.33 Using Preemption

PBS provides the ability to preempt currently running jobs in order to run higher-priority work. This is called preemption or preemptive scheduling. Preemption follows a set of preemption priorities that you define. Jobs that have high preemption priority preempt those with low preemption priority. Preemption priority is mostly independent of execution priority.

Preemption is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

4.8.33.1 Terminology

- Preempt
  Stop one or more running jobs in order to start a higher-priority job

- Preemption level
  Job characteristic that determines preemption priority. Levels can be things like being in an express queue, starving, having an owner who is over a soft limit, being a normal job, or having an owner who is over a fairshare allotment

- Preemption method
  The method by which a job is preempted. This can be checkpointing, suspension, or requeueing

- Preemption priority
  How important this job is compared to other jobs, when considering whether to preempt

4.8.33.2 Preemption Parameters

The scheduler parameters that control preemption are defined in PBS_HOME/sched_priv/sched_config. They are listed here:
Scheduling

Chapter 4

**preemptive_sched**
Enables job preemption.
Format: *String*
Default: *True all*

**preempt_order**
Defines the order of preemption methods which the scheduler will use on jobs.
Format: *String*, as quoted list
Default: "SCR"

**preempt_prio**
Specifies the ordering of priority of different preemption levels.
Format: *String*, as quoted list
Default: "express_queue, normal_jobs"

**preempt_queue_prio**
Specifies the minimum queue priority required for a queue to be classified as an express queue.
Format: *Integer*
Default: 150

**preempt_sort**
Whether jobs most eligible for preemption will be sorted according to their start times. Allowable values: "min_time_since_start", or no preempt_sort setting. If set to "min_time_since_start", first job preempted will be that with most recent start time. If not set, preempted job will be that with longest running time. Must be commented out in order to be unset; default scheduler configuration file has this parameter set to *min_time_since_start*.
Format: *String*
Default: *min_time_since_start*

### 4.8.33.3 How Preemption Works

If preemption is enabled, the scheduler uses preemption as part of its normal pattern of examining each job and figuring out whether or not it can run now. If a job with high preemption priority cannot run immediately, the scheduler looks for jobs with lower preemption priority. The scheduler preempts these jobs and uses their resources for the higher-priority job. The scheduler tracks resources used by lower-priority jobs, looking for enough resources to run the higher-priority job. If the scheduler cannot find enough work to preempt in order to run a given job, it will not preempt any work.

A job running in a reservation cannot be preempted.
A job’s preemption priority is determined by its preemption level.

4.8.33.4 Triggers for Preemption

If preemption is enabled, preemption is used during the following:

- The normal scheduling cycle
- When you run a job via `qrun`

4.8.33.5 Preemption Levels

A preemption level is a class of jobs, where all the jobs in the class share a characteristic. PBS provides built-in preemption levels, listed in the table below.

<table>
<thead>
<tr>
<th>Preemption Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>express_queue</td>
<td>Jobs in express queues. See section 4.8.33.5.ii, “The Express Queues Preemption Level”, on page 233</td>
</tr>
<tr>
<td>starving_jobs</td>
<td>A job that is starving. See section 4.8.33.5.iv, “The Starving Job Preemption Level”, on page 234</td>
</tr>
<tr>
<td>fairshare</td>
<td>When the entity owning a job exceeds its fairshare limit. See section 4.8.33.5.iii, “The Fairshare Preemption Level”, on page 234</td>
</tr>
<tr>
<td>queue_softlimits</td>
<td>Jobs which are over their queue soft limits. See section 4.8.33.5.i, “The Soft Limits Preemption Level”, on page 231</td>
</tr>
<tr>
<td>server_softlimits</td>
<td>Jobs which are over their server soft limits. See section 4.8.33.5.i, “The Soft Limits Preemption Level”, on page 231</td>
</tr>
<tr>
<td>normal_jobs</td>
<td>The preemption level into which a job falls if it does not fit into any other specified level. See section 4.8.33.5.v, “The Normal Jobs Preemption Level”, on page 234</td>
</tr>
</tbody>
</table>

You can specify the relative priority of each preemption level, by listing the levels in the desired order in the `preempt_prio` scheduler parameter. Placing a level earlier in the list, meaning to the left, gives it higher priority. For example, if your list is “express_queue”, “normal_jobs”, “server_softlimits”, you are giving the highest priority to jobs in express queues, and the lowest priority to jobs that are over their
server soft limits. You can list levels in any order, but be careful not to work at cross-purposes with your execution priority. See section 4.8.16, “Calculating Job Execution Priority”, on page 166.

The default value for preempt_prio is the following:

```plaintext
preempt_prio: “express_queue, normal_jobs”
```

If you do not list a preemption level in the preempt_prio scheduler parameter, the jobs in that level are treated like normal jobs. For example, if you do not list server_softlimits, then jobs that are over their server soft limits are treated like jobs in the normal_jobs level.

You can create new levels, which are combinations of the built-in levels. For example, you can define a level which is `express_queue + server_softlimits`. This level contains jobs that are in express queues and are over their server soft limits. You would probably want to place this level just to the right of the `express_queue` level, meaning that these jobs could be preempted by jobs that are in express queues but are not over their server soft limits.

You can give two or more levels the same priority. To do this, put a plus sign (“+”) between them, and do not list either level separately in preempt_prio. You are creating a new level that includes all the built-in levels that should have the same priority. For example, to list express queue jobs as highest in priority, then fairshare and starving jobs at the next highest priority, then normal jobs last, create a new level that contains the fairshare and starving_jobs levels:

```plaintext
preempt_prio: “express_queue, fairshare+starving_jobs, normal_jobs”
```

You must list normal_jobs in the preempt_prio scheduler parameter.

### 4.8.33.5.i The Soft Limits Preemption Level

You can set a limit, called a hard limit, on the number of jobs that can be run or the amount of a resource that can be consumed by a person, a group, or by everyone, and this limit can be applied at the server and at each queue. If you set such a limit, that is the greatest number of jobs that will be run, or the largest amount of the resource that will be consumed.

You can also set a soft limit on the number of jobs that can be run or the amount of a resource that can be consumed. This soft limit should be lower than the hard limit, and should mark the point where usage changes from being normal to being “extra, but acceptable”. Usage in this “extra, but acceptable” range can be treated by PBS as being lower priority than the nor-
mal usage. PBS can preempt jobs that are over their soft limits. The difference between the soft limit and the hard limit provides a way for users or groups to use resources as long as no higher-priority work is waiting.

Example 4-17: Using group soft limits

One group of users, group A, has submitted enough jobs that the group is over their soft limit. A second group, group B, submits a job and are under their soft limit. If preemption is enabled, jobs from group A are preempted until the job from group B can run.

Example 4-18: Using soft limits on number of running jobs

Given the following:
- You have three users, UserA, UserB, and UserC
- Each has a soft limit of 3 running jobs
- UserA runs 3 jobs
- UserB runs 4 jobs
- UserC submits a job to an express queue

This means:
- UserB has 1 job over the soft limit, so UserB’s jobs are eligible for preemption by UserC’s job

Example 4-19: Using soft limits on amount of resource being used

Given the following:
- Queue soft limit for ncpus is 8
- UserA’s jobs use 6 CPUs
- UserB’s jobs use 10 CPUs

This means:
- UserB is over their soft limit for CPU usage
- UserB’s jobs are eligible for preemption

To use soft limits in preemption levels, you must define soft limits. Soft limits are specified by setting server and queue limit attributes. The attributes that control soft limits are:

- `max_run_soft`
  Sets the soft limit on the number of jobs that can be running

- `max_run_res_soft.<resource>`
  Sets the soft limit on the amount of a resource that can be consumed by running jobs

Soft limits are enforced only when they are used as a preemption level.
To use soft limits as preemption levels, add their keywords to the `preempt_prio` parameter in the scheduler’s configuration file:

- To create a preemption level for those over their soft limits at the server level, add “server_softlimits” to the `preempt_prio` parameter.
- To create a preemption level for those over their soft limits at the queue level, add “queue_softlimits” to the `preempt_prio` parameter.
- To create a preemption level for those over their soft limits at both the queue and server, add “server_softlimits+queue_softlimits” to the `preempt_prio` parameter.

The jobs of a user or group are over their soft limit only as long as the number of running jobs or the amount of resources used by running jobs is over the soft limit. If some of these jobs are preempted or finish running, and the soft limit is no longer exceeded, the jobs of that user or group are no longer over their soft limit, and no longer in that preemption level. For example, if the soft limit is 3 running jobs, and UserA runs 4 jobs, as soon as one job is preempted and only 3 of UserA’s jobs are running, UserA’s jobs are no longer over their soft limit.

For a complete description of the use of these attributes, see section 5.15.1.4, “Hard and Soft Limits”, on page 371.

### 4.8.33.5.ii The Express Queues Preemption Level

The `express_queue` preemption level applies to jobs residing in express queues. An express queue is an execution queue with priority at or above the value set in the `preempt_queue_prio` scheduler parameter. The default value for this parameter is 150.

Express queues do not require the `by_queue` or `sort_queues` scheduler parameters to be `True`.

If you will use the `express_queue` preemption level, you probably want to configure at least one express queue, along with some method of moving jobs into it. See section 2.2, “Queues”, on page 18.

If you have more than one express queue, and they have different priorities, you are effectively creating separate sub-levels for express queues. Jobs in a higher-priority express queue have greater preemption priority than jobs in lower-priority express queues.

See “`preempt_queue_prio`” on page 291 of the PBS Professional Reference Guide.
4.8.33.5.iii  The Fairshare Preemption Level

The fairshare preemption level applies to jobs owned by entities who are over their fairshare allotment. For example, if each of five users has 20 percent of the fairshare tree, and UserA is using 25 percent of the resources being tracked for fairshare, UserA’s jobs become eligible for preemption at the fairshare preemption level.

To use the fairshare preemption level, you must enable fairshare. See section 4.8.18, “Using Fairshare”, on page 170.

4.8.33.5.iv  The Starving Job Preemption Level

The starving_jobs preemption level applies to jobs that are starving. Starving jobs are jobs that have been waiting at least a specified amount of time to run.

To use the starving_jobs preemption level, you must enable starving:

- Set the PBS_HOME/sched_priv/sched_config help_starving_jobs parameter to True
- Set the amount of time that a job must wait before it is starving in the max_starve scheduler parameter


4.8.33.5.v  The Normal Jobs Preemption Level

One special class, normal_jobs, is the default class for any job not otherwise specified. If a job does not fall into any of the specified levels, it is placed in normal_jobs.

Example 4-20: Starving jobs have the highest priority, then normal jobs, then jobs whose entities are over their fairshare limit:

    preempt_prio: “starving_jobs, normal_jobs, fairshare”

Example 4-21: Starving jobs whose entities are also over their fairshare limit are lower priority than normal jobs:

    preempt_prio: “normal_jobs, starving_jobs+fairshare”
4.8.33.6 Selecting Preemption Level

PBS places each job in the most exact preemption level, or the highest preemption level that fits the job.

Example 4-22: We have a job that is starving and over its server soft limits. The job is placed in the “starving_jobs” level:

```
preempt_prio: “starving_jobs, normal_jobs, server_softlimits”
```

Example 4-23: We have a job that is starving and over its server soft limits. The job is placed in the “starving_jobs+server_softlimits” level:

```
preempt_prio: “starving_jobs, starving_jobs+server_softlimits, normal_jobs, server_softlimits”
```

4.8.33.7 Sorting Within Preemption Level

If there is more than one job within the preemption level chosen for preemption, PBS chooses jobs within that level according to their start time. By default, PBS preempts the job which started running most recently. This behavior can be changed using the scheduler parameter `preempt_sort`. To direct PBS to preempt the longest-running jobs, comment out the line containing the `preempt_sort` parameter in `PBS_HOME/sched_priv/sched_config`.

For example, if we have two jobs where job A started running at 10:00 a.m. and job B started running at 10:30 a.m:

- The default behavior preempts job B
- Job A is preempted when `preempt_sort` is commented out

The allowable value for the `preempt_sort` parameter is “min_time_since_start”.

The default value for the `preempt_sort` parameter is “min_time_since_start”. Must be commented out in order to be unset; default scheduler configuration file has this parameter set to `min_time_since_start`.

4.8.33.8 Preemption Methods

The scheduler can preempt a job in one of the following ways:

- Suspend the job
- Checkpoint the job
- Requeue the job
The scheduler tries to preempt a job using the methods listed in the order you specify. This means that if you specify that the order is “checkpoint, suspend, requeue”, the scheduler first tries to checkpoint the job, and if it cannot, it tries to suspend the job, and if it cannot do that, it tries to requeue the job.

You can specify the order of these attempts in the `preempt_order` scheduler parameter in `PBS_HOME/sched_priv/sched_config`.

The `preempt_order` parameter defines the order of preemption methods which the scheduler uses on jobs. This order can change depending on the percentage of time remaining on the job. The ordering can be any combination of `S`, `C` and `R` (for suspend, checkpoint, and requeue).

The contents is an ordering, for example “SCR” optionally followed by a percentage of time remaining and another ordering.

The format is a quoted list (“”).

Example 4-24: PBS should first attempt to use suspension to preempt a job, and if that is unsuccessful, then requeue the job:

```
preempt_order: “SR”
```

Example 4-25: If the job has between 100-81% of requested time remaining, first try to suspend the job, then try checkpoint, then requeue. If the job has between 80-51% of requested time remaining, then attempt suspend then checkpoint; and between 50% and 0% time remaining just attempt to suspend the job:

```
preempt_order: “SCR 80 SC 50 S”
```

The default value for `preempt_order` is “SCR”.

### 4.8.33.8.i Preemption Via Checkpoint

When a job is preempted via checkpointing, MoM runs the `checkpoint_abort` script, and PBS kills and requeues the job. When the scheduler elects to run the job again, the scheduler runs the restart script to restart the job from where it was checkpointed.

To preempt via checkpointing, you must define both of the following:

- The checkpointing action in the MoM’s `checkpoint_abort $action` parameter that is to take place when the job is preempted
- The restarting action in the MoM’s `restart $action` parameter that is to take place when the job is restarted

To do this, you must supply checkpointing and restarting scripts or equivalents, and then configure the MoM’s `checkpoint_abort` and `restart $action` parameters. Do not use the `$action` parameter when the job should keep running.

4.8.33.8.ii Preemption Via Suspension

Jobs are normally suspended via the SIGSTOP signal and resumed via the SIGCONT signal. An alternate suspend or resume signal can be configured in MoM’s $suspendsig configuration parameter. See “pbs_mom” on page 58 of the PBS Professional Reference Guide.

4.8.33.8.iii Suspended Jobs and PBS Licenses

When a job is suspended, its PBS usage licenses are returned to the license pool, subject to the constraints of the server’s pbs_license_min and pbs_license_linger_time attributes. The scheduler checks to make sure that licenses are available before resuming any job. If the required licenses are not available, the scheduler will log a message and add a comment to the job. See “Floating Licenses and Job States” on page 106 in the PBS Professional Installation & Upgrade Guide.

4.8.33.8.iv Suspended Jobs and Resources

Suspended jobs will hold onto some memory and disk space. Suspended jobs may hold application licenses if the application releases them only when it exits. See section 5.9.6.2.i, “Suspension/resumption Resource Caveats”, on page 314.

4.8.33.8.v Preemption Via Requeue

When a job is preempted and requeued, the job stops execution and is requeued. A requeued job’s eligible time is preserved. The amount of time allowed to requeue a job is controlled by the job_requeue_timeout server attribute. See “Server Attributes” on page 316 of the PBS Professional Reference Guide.

A job that is not eligible to be requeued, meaning a job that was submitted with “-r n”, will not be selected to be preempted via requeue.

4.8.33.9 Enabling Preemption

Preemptive scheduling is enabled by setting parameters in the scheduler’s configuration file PBS_HOME/sched_priv/sched_config.
To enable preemption, you must do the following:

1. Specify the preemption levels to be used by setting `preempt_prio` to desired preemption levels (the default is “express_queue, normal_jobs”).
   
The `preempt_prio` parameter must contain an entry for `normal_jobs`.

2. Optional: specify preemption order by setting `preempt_order`.

3. Optional: specify whether longest- or shortest-running jobs should be preempted first by setting `preempt_sort`.

4. If you will use the `fairshare` preemption level, configure fairshare. See section 4.8.18, “Using Fairshare”, on page 170.

5. If you will use the `starving_jobs` preemption level, configure starving. See section 4.8.33.5.iv, “The Starving Job Preemption Level”, on page 234.

6. If you will use the `server_softlimits` and/or `queue_softlimits` preemption levels, configure server and/or queue soft limits. See section 4.8.33.5.i, “The Soft Limits Preemption Level”, on page 231.

7. Enable preemption by setting `preemptive_sched` to `True`. It is `True` by default.

8. Choose whether to use preemption during primetime, non-primetime, or all of the time. The default is `ALL`. If you want separate behavior for primetime and non-primetime, specify each separately. For example:
   
   ```
   preemptive_sched True prime
   preemptive_sched False non_prime
   ```

9. If you are sorting jobs on a key using `job_sort_key`, make `preempt_priority` the primary sort key:
   
   ```
   job_sort_key: “preempt_priority HIGH”
   ```
4.8.33.10 Preemption Example

Below is an example of (part of) the scheduler’s configuration file, showing an example configuration for preemptive scheduling.

```bash
# turn on preemptive scheduling
#
preemptive_sched: TRUE ALL
#
# set the queue priority level for express queues
#
preempt_queue_prio: 150
#
# specify the priority of jobs as: express queue
# (highest) then starving jobs, then normal jobs,
# followed by jobs who are starving but the user/group
# is over a soft limit, followed by users/groups over
# their soft limit but not starving
#
preempt_prio: "express_queue, starving_jobs, normal_jobs,
  starving_jobs+server_softlimits, server_softlimits"
#
# specify when to use each preemption method.
# If the first method fails, try the next
# method. If a job has between 100-81% time
# remaining, try to suspend, then checkpoint
# then requeue. From 80-51% suspend and then
# checkpoint, but don’t requeue.
# If between 50-0% time remaining, then just
# suspend it.
#
preempt_order: "SCR 80 SC 50 S"
```

4.8.33.11 Preemption Caveats and Recommendations

- When using any of the fairshare, soft limits, express queue, or starving jobs preemption levels, be sure to enable the corresponding PBS feature. For example, when using preemption with the fairshare preemption level, be sure to turn fairshare on. Otherwise, you
will be using stale fairshare data to preempt jobs.

- It’s important to be careful about the order of the preemption levels and the sizes of the limits at queue and server. For example, if you make users who are over their server soft limits have higher priority than users who are over their queue soft limits, and you set the soft limit higher at the server than at the queue, you can end up with users who have more jobs running preempting users who have fewer jobs running.

In this example, a user with more jobs preempts a user with fewer jobs

Given the following:
- preempt_prio line contains “server_softlimits, queue_softlimits”
- Server soft limit is 5
- Queue soft limit is 3
- User1 has 6 jobs running
- User2 has 4 jobs running

This means:
- User1 has higher priority, because User1 is over the server soft limit
- User1’s jobs can preempt User2’s jobs

To avoid this scenario, you could set the preempt_prio line to contain “server_softlimits, queue_softlimits, server_softlimits+queue_softlimits”. In this case User1 would have lower priority, because User1 is over both soft limits.

- Preemption priority is independent of execution priority. You can list preemption levels in any order in preempt_prio, but be careful not to work at cross-purposes with your execution priority. Be sure that you are not preempting jobs that have higher execution priority. See section 4.8.16, “Calculating Job Execution Priority”, on page 166.

- If you are sorting jobs using job_sort_key, we recommend that when you change preempt_prio from the default, you add preempt_priority as the primary key for the job_sort_key scheduler parameter. This is especially important when using soft limits for preemption. Jobs that are over their soft limits should have lower priority, both in execution and preemption. For example, if you’re using soft limit preemption levels, but not assigning job execution priority according to soft limits, you can have job A started and preempted in the same scheduler cycle, because job A has higher execution priority, but is over its soft limits, and has lower preemption priority. To sort jobs on preemption priority:
  job_sort_key: “preempt_priority HIGH”

  See section 4.8.43, “Sorting Jobs on a Key”, on page 275.

- Using preemption with strict ordering and backfilling may change which job is being
backfilled around.

• When a job is suspended via checkpoint or requeue, it loses its queue wait time. This does not apply to preemption via suspension.

• If a high-priority job has been selected to preempt lower-priority jobs, but is rejected by a runjob hook, the scheduler undoes the preemption of the low-priority jobs. Suspended jobs are resumed, and checkpointed jobs are restarted.

• A job that has requested an AOE will not preempt another job, regardless of whether the job’s requested AOE matches an instantiated AOE. Running jobs are not preempted by jobs requesting AOEs.

• If a job is checkpointed by the scheduler because it was preempted, the scheduler briefly applies a hold, but releases the hold immediately after checkpointing the job, and runs the restart script when the job is scheduled to run.

• When jobs are preempted via requeueing, the requeue can fail if the job being preempted takes longer than the allowed timeout. See section 9.4.3, “Setting Job Requeue Timeout”, on page 737.

• A job that is suspended has higher execution priority than a queued job. Therefore, a job that is preempted via suspension has higher priority than a job preempted via checkpoint or requeue, because those jobs are in the queued state.

• When you issue “qrun <job ID>”, without the -H option, the selected job has preemption priority between Reservation and Express, for that scheduling cycle. However, at the following scheduling cycle, the preemption priority of the selected job returns to whatever it would be without qrun.

4.8.34 Using Primetime and Holidays

Often is it useful to run different scheduling policies for specific intervals during the day or work week. PBS provides a way to specify two types of interval, called *primetime* and *non-primetime*.

Between them, primetime and non-primetime cover all time. There is no time slot that is neither primetime nor non-primetime. This includes dedicated time. Primetime and/or non-primetime overlap dedicated time.

You can use non-primetime for such tasks as running jobs on desktop clusters at night.

4.8.34.1 How Primetime and Holidays Work

The scheduler looks in the PBS_HOME/sched_priv/holidays file for definitions of primetime, non-primetime, and holidays.
Many PBS scheduling parameters can be specified separately for primetime, non-primetime, or all of the time. This means that you can use, for example, fairshare during primetime and no fairshare during non-primetime. These parameters have a time slot default of all, meaning that if enabled, they are in force all of the time.

The scheduler applies the parameters defined for primetime during the primetime time slots, and applies parameters defined for non-primetime during the non-primetime time slots. Any scheduler parameters defined for all time are run whether it is primetime or not.

Any holidays listed in the holidays file are treated as non-primetime. To have a holiday treated like a normal workday or weekend, do not list it in the holidays file.

There are default behaviors for primetime and non-primetime, but you can set up the behavior you want for each type. The names “primetime” and “non-primetime” are meant to be informative, but they are arbitrary. The default for primetime is from 6:00 AM to 5:30 PM on weekdays, meaning that weekends and nights are non-primetime by default. U.S. Federal holidays are provided in the holidays file.

You can define primetime and non-primetime queues. Jobs in these queues can run only during the designated time. Queues that are not defined specifically as primetime or non-primetime queues are called “anytime queues”.

### 4.8.34.2 Configuring Primetime and Non-primetime

In order to use primetime and non-primetime, you must have a `holidays` file with the current year in it.

You can specify primetime and non-primetime time slots by specifying them in the `PBS_HOME/sched_priv/holidays` file.

The format of the primetime and non-primetime section of the `holidays` file is the following:

```
YEAR YYYY
<day> <prime> <nonprime>
<day> <prime> <nonprime>
```

If there is no `YEAR` line in the holidays file, primetime is in force at all times. If there is more than one `YEAR` line, the last one is used.

In `YEAR YYYY, YYYY` is the current year.

*Day* can be *weekday, monday, tuesday, wednesday, thursday, friday, saturday, or sunday.*

Each line must have all three fields.

Any line that begins with a “*” or a “#” is a comment.
Weekday names must be lowercase.

The ordering of elements in this file is important. The ordering of <day> lines in the holidays file controls how primetime is determined. A later line takes precedence over an earlier line.

For example:

```
weekday  0630  1730
friday   0715  1600
```

means the same as

```
monday   0630  1730
tuesday  0630  1730
wednesday 0630  1730
thursday 0630  1730
friday   0715  1600
```

However, if a specific day is followed by “weekday”,

```
friday   0700  1600
weekday  0630  1730
```

the “weekday” line takes precedence, so Friday will have the same primetime as the other weekdays.

Times can be expressed as one of the following:

- **HHMM** with no colons(:)
- The word “all”
- The word “none”

### 4.8.34.3 Configuring Holidays

You can specify primetime and non-primetime time slots by specifying them in the PBS_HOME/sched_priv/holidays file.

You must specify the year, otherwise primetime is in force at all times, and PBS will not recognize any holidays. Specify the year here, where YYYY is the current year:

```
YEAR YYYY
```

Holidays are specified in lines of this form:

```
<day of year> <month day-of-month> <holiday name>
```

PBS uses the <day of year> field and ignores the <date> string.

*Day of year* is the julian day of the year between 1 and 365 (e.g. “1”).
Month day-of-month is the calendar date, for example “Jan 1”.

Holiday name is the name of the holiday, for example “New Year’s Day”.

### 4.8.34.4 Example of holidays File

```
YEAR  2007
*     Prime   Non-Prime
* Day  Start   Start
* weekday  0600   1730
   saturday none   all
   sunday  none   all
*
* Day of Calendar     Company Holiday
* Year  Date   Holiday
   1      Jan 1    New Year's Day
   15     Jan 15   Dr. M.L. King Day
   50     Feb 19   President's Day
   148    May 28   Memorial Day
   185    Jul  4   Independence Day
   246    Sep  3   Labor Day
   281    Oct  8   Columbus Day
   316    Nov 12   Veteran's Day
   326    Nov 22   Thanksgiving
   359    Dec 25   Christmas Day
```

### 4.8.34.5 Reference Copies of holidays File

Reference copies of the holidays file are provided in PBS_EXEC/etc/holidays.<year>.

The current year’s holidays file has a reference copy in PBS_EXEC/etc/pbs_holidays, and a copy used by PBS in PBS_HOME/sched_priv/holidays.

To use a particular year’s file as the holidays file, copy it to PBS_HOME/sched_priv/holidays -- note the “s” on the end of the filename.
4.8.34.6 Defining Primetime and Non-primetime Queues

Jobs in a primetime queue can start only during primetime. Jobs in a non-primetime queue can start only during non-primetime. Jobs in an anytime queue can start at any time.

You define a primetime queue by naming it using the primetime prefix. The prefix is defined in the `primetime_prefix` scheduler parameter. The default is "p_". For example, you could name a primetime queue "p_queueA", using the default.

Similarly, you define a non-primetime queue by prefixing the name. The prefix is defined in the `nonprimetime_prefix` scheduler parameter, and defaults to "np_".

4.8.34.7 Controlling Whether Jobs Cross Primetime Boundaries

You can control whether jobs are allowed to start running in one time slot and finish in another, for example when job A starts during primetime and finishes a few minutes into non-primetime. When a job runs past the boundary, it delays the start of a job that is constrained to run only in the later time slot. For example, if job B can run only during non-primetime, it may have to wait while job A uses up non-primetime before it can start. You can control this behavior for all queues, or you can exempt anytime queues, controlling only primetime and non-primetime queues. You can also specify how much time past the boundary a job is allowed to run.

To prevent the scheduler from starting any jobs which would run past a primetime/non-primetime boundary, set the `backfill_prime` scheduler parameter to `True`. You can specify this separately for primetime and non-primetime. If you specify it for one type of time slot, it prevents those jobs from crossing the next boundary. For example, if you set the following:

```
backfill_prime True prime
```

jobs in primetime slots are not allowed to cross into non-primetime slots.

If you set the following:

```
backfill_prime True non_prime
```

jobs in non-primetime slots are not allowed to cross into primetime slots.

To exempt jobs in anytime queues from the control of `backfill_prime`, set the `prime_exempt_anytime_queues` scheduler parameter to `True`. This means that jobs in an anytime queue are not prevented from running across a primetime/non-primetime or non-primetime/primetime boundary.
Chapter 4  Scheduling

To allow jobs to spill over a certain amount of time past primetime/non-primetime boundaries, but no more, specify this amount of time in the prime_spill scheduler parameter. You can specify separate behavior for primetime and non-primetime jobs. For example, to allow primetime jobs to spill by 20 minutes, but only allow non-primetime jobs to spill by 1 minute:

prime_spill 00:20:00 prime
prime_spill 00:01:00 non_prime

The prime_spill scheduler parameter applies only when backfill_prime is True.

4.8.34.8 Logging

The scheduler logs a message at the beginning of each scheduling cycle indicating whether it is primetime or not, and when this period of primetime or non-primetime will end. The message is at log event class 0x0100. The message is of this form:

"It is primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS"
or

"It is non-primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS"

4.8.34.9 Scheduling Parameters Affecting Primetime

backfill_prime
The scheduler will not run jobs which would overlap the boundary between primetime and non-primetime.
Format: Boolean
Default: False

nonprimetime_prefix
Queue names which start with this prefix will be treated as non-primetime queues. Jobs within these queues will only run during non-primetime.
Format: String
Default: np_

primetime_prefix
Queue names starting with this prefix are treated as primetime queues. Jobs will only run in these queues during primetime.
Format: String
Default: p_

prime_exempt_anytime_queues
Determines whether anytime queues are controlled by backfill_prime.
If set to *True*, jobs in an anytime queue will not be prevented from running across a primetime/non-primetime or non-primetime/primetime boundary.

If set to *False*, the jobs in an anytime queue may not cross this boundary, except for the amount specified by their `prime_spill` setting.

Format: `Boolean`
Default: `False`

`prime_spill`

Specifies the amount of time a job can spill over from non-primetime into primetime or from primetime into non-primetime. This option can be separately specified for prime- and non-primetime. This option is only meaningful if `backfill_prime` is *True*.

Format: `Duration`
Default: `00:00:00`

### 4.8.34.10 Primetime and Holiday Caveats

- In order to use primetime and non-primetime, you must have a `holidays` file with the current year in it. If there is no `holidays` file with a year in it, primetime is in force all of the time.

- You cannot combine `holidays` files.

- If you use the formula, it is in force all of the time.

### 4.8.35 Provisioning

PBS provides automatic provisioning of an OS or application, on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application.

You can configure vnodes so that PBS will automatically install the OS or application that jobs need in order to run on those vnodes. For example, you can configure a vnode that is usually running RHEL to run SLES instead whenever the Physics group runs a job requiring SLES. If a job requires an application that is not usually installed, PBS can install the application in order for the job to run.

You can use provisioning for booting multi-boot systems into the desired OS, downloading an OS to and rebooting a diskless system, downloading an OS to and rebooting from disk, instantiating a virtual machine, etc. You can also use provisioning to run a configuration script or install an application.
4.8.36 Queue Priority

Queues and queue priority play several different roles in scheduling, so this section contains pointers to other sections.

Each queue can have a different priority. A higher value for priority means the queue has greater priority. By default, queues are sorted from highest to lowest priority. Jobs in the highest priority queue will be considered for execution before jobs from the next highest priority queue. If queues don’t have different priority, queue order is undefined.

Each queue’s priority is specified in its priority attribute. By default, the queue priority attribute is unset. There is no limit to the priority that you can assign to a queue, however it must fit within integer size. See “Queue Attributes” on page 354 of the PBS Professional Reference Guide.

4.8.36.1 Configuring Queue Priority

You can specify the priority of each queue by setting a value for its priority attribute:

```
Qmgr: set queue <queue name> priority = <value>
```

4.8.36.2 Using Queue Priority

You can configure the scheduler so that job execution or preemption priority is partly or entirely determined by the priority of the queue in which the job resides. Queue priority can be used for the following purposes:

- Queue priority can be used as a term in the job sorting formula. See section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.
- Queue priority can be used to specify order in which queues are examined when scheduling jobs. If you want jobs to be examined queue by queue, in order of queue priority, you must also sort the queues and specify a different priority for each queue. A queue with a higher value is examined before a queue with a lower value. See section 4.2.5.3.i, “Using Queue Order to Affect Order of Consideration”, on page 68.
- You can set up execution priority levels that include jobs in express queues. For information on configuring job priorities in the scheduler, see section 4.8.16, “Calculating Job Execution Priority”, on page 166.
- You can set up preemption levels that include jobs in express queues. For information on preemption, see section 4.8.33, “Using Preemption”, on page 228.
A queue is an express queue if its priority is greater than or equal to the value that defines an express queue. For more about using express queues, see section 4.8.17, “Express Queues”, on page 169.

4.8.36.3 Queue Priority Caveats

• If you use queue priority in the formula and the job is moved to another server through peer scheduling, the queue priority used in the formula will be that of the new queue to which the job is moved.

4.8.37 Advance and Standing Reservations

PBS provides a way to reserve specific resources for a defined time period. You can make a one-time reservation, or you can make a series of reservations, where each one is for the same resources, but for a different time period.

Reservations are useful for accomplishing the following:

• To get a time slot on a specific host
• To run a job in a specific time slot, meaning at or by a specific time
• To be sure a job will run
• To have a high-priority job run soon

4.8.37.1 Definitions

Advance reservation
A reservation for a set of resources for a specified time. The reservation is available only to the creator of the reservation and any users or groups specified by the creator.

Standing reservation
An advance reservation which recurs at specified times. For example, the user can reserve 8 CPUs and 10GB every Wednesday and Thursday from 5pm to 8pm, for the next three months.

Occurrence of a standing reservation
An occurrence of a standing reservation behaves like an advance reservation, with the following exceptions:

• While a job can be submitted to a specific advance reservation, it can only be submitted to the standing reservation as a whole, not to a specific occur-
You can only specify when the job is eligible to run. See “qsub” on page 211 of the PBS Professional Reference Guide.

- When an advance reservation ends, it and all of its jobs, running or queued, are deleted, but when an occurrence ends, only its running jobs are deleted.

Each occurrence of a standing reservation has reserved resources which satisfy the resource request, but each occurrence may have its resources drawn from a different source. A query for the resources assigned to a standing reservation will return the resources assigned to the soonest occurrence, shown in the resv_nodes attribute reported by pbs_rstat.

Also called an instance of a standing reservation.

**Soonest occurrence of a standing reservation**

The occurrence which is currently active, or if none is active, then it is the next occurrence.

**Degraded reservation**

An advance reservation for which one or more associated vnodes are unavailable.

A standing reservation for which one or more vnodes associated with the soonest occurrence are unavailable.

### 4.8.37.2 How Reservations Work

#### 4.8.37.2.i Creating Reservations

Any PBS user can create both advance and standing reservations using the `pbs_rsub` command. PBS either confirms that the reservation can be made, or rejects the request. Once the reservation is confirmed, PBS creates a queue for the reservation’s jobs. Jobs are then submitted to this queue.

When a reservation is confirmed, it means that the reservation will not conflict with currently running jobs, other confirmed reservations, or dedicated time, and that the requested resources are available for the reservation. A reservation request that fails these tests is rejected. All occurrences of a standing reservation must be acceptable in order for the standing reservation to be confirmed.

The `pbs_rsub` command returns a reservation ID, which is the reservation name. For an advance reservation, this reservation ID has the format:

\[R<\text{unique integer}>.<\text{server name}>\]

For a standing reservation, this reservation ID refers to the entire series, and has the format:

\[S<\text{unique integer}>.<\text{server name}>\]
The user specifies the resources for a reservation using the same syntax as for a job. See "Reserving Resources Ahead of Time", on page 185 of the PBS Professional User’s Guide, for detailed information on creation and use of reservations.

The time for which a reservation is requested is in the time zone at the submission host.

4.8.37.2.ii Reservations and Placement Sets

When PBS chooses a placement set for a reservation, it makes the same choices as it would for a regular job. It fits the reservation into the smallest possible placement set. See section 4.8.32.4.ii, “Order of Placement Set Consideration Within Pool”, on page 215.

When a reservation is created, it is created within a placement set, if possible. If no placement set will satisfy the reservation, placement sets are ignored. The vnodes allocated to a reservation are used as one single placement set for jobs in the reservation; they are not subdivided into smaller placement sets. A job within a reservation runs within the single placement set made up of the vnodes allocated to the reservation.

4.8.37.2.iii Requesting Resources for Reservations

Reservations request resources using the same mechanism that jobs use. If a resource is unrequestable, users cannot request it for a reservation. If a resource is invisible, users cannot view it or request it for a reservation.

4.8.37.2.iv Reservations and Provisioning

Users can create reservations that request AOE. Each reservation can have at most one AOE specified for it. Any jobs that run in that reservation must not request a different AOE. See section 7.4.3, “Provisioning And Reservations”, on page 603.

The vnodes allocated to a reservation that requests an AOE are put in the resv-exclusive state when the reservation runs. These vnodes are not shared with other reservations or with jobs outside the reservation.

For information on restrictions applying to reservations used with provisioning, see section 7.7.2.3, “Vnode Reservation Restrictions”, on page 621.

For how to avoid problems with provisioning and reservations, see section 7.10.1, “Using Provisioning Wisely”, on page 637.

4.8.37.2.v Reservation Priority

A job running in a reservation cannot be preempted.

A job running in a reservation has the highest execution priority.
4.8.37.3 Querying Reservations

To query a reservation, use the `pbs_rstat` command. See "Viewing the Status of a Reservation", on page 192 of the PBS Professional User’s Guide. To delete an advance reservation, use the `pbs_rdel` command, not the `qmgr` command.

4.8.37.4 Controlling Access to Reservations

You can specify which users and groups can and cannot submit jobs to reservations. Use the `qmgr` command to set the reservation queue’s `acl_users` and/or `acl_groups` attributes. See section 8.3, “Using Access Control”, on page 647.

4.8.37.5 Reservation Fault Tolerance

PBS automatically keeps track of the vnodes assigned to reservations, and tries to find replacement vnodes for those that become unavailable. See section 9.5, “Reservation Fault Tolerance”, on page 743.

4.8.37.6 Advance and Standing Reservations and Licensing

Reservation jobs won’t run if PBS runs out of licenses. Set the server’s `pbs_license_min` attribute to the total number of CPUs, including virtual CPUs, in the PBS complex. See “Floating Licenses and Reservations” on page 107 in the PBS Professional Installation & Upgrade Guide and “Setting Server Licensing Attributes” on page 96 in the PBS Professional Installation & Upgrade Guide.

4.8.37.7 Logging Reservation Information

The start and end of each occurrence of a standing reservation is logged as if each occurrence were a single advance reservation.

Reservation-related messages are logged at level 0x0200 (512).

4.8.37.8 Accounting

Resources requested for a reservation are recorded in the reservation’s `Resource_List` attribute, and reported in the accounting log B record for the reservation. The accounting log B record is written at the beginning of a reservation.
4.8.37.9 Attributes Affecting Reservations

Most of the attributes controlling a reservation are set when the reservation is created by the user. However, some server and vnode attributes also control the behavior of reservations. The server attributes that affect reservations are listed here, and described in “Server Attributes” on page 316 of the PBS Professional Reference Guide.

Table 4-14: Server Attributes Affecting Reservations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl_resv_host_enable</td>
<td>Controls whether or not the server uses the acl_resv_hosts access control lists.</td>
</tr>
<tr>
<td>acl_resv_hosts</td>
<td>List of hosts from which reservations may and may not be created at this server.</td>
</tr>
<tr>
<td>acl_resv_group_enable</td>
<td>Controls whether or not the server uses the acl_resv_groups access control lists.</td>
</tr>
<tr>
<td>acl_resv_groups</td>
<td>List of groups who may and may not create reservations at this server.</td>
</tr>
<tr>
<td>acl_resv_user_enable</td>
<td>Controls whether or not the server uses the acl_resv_users access control lists.</td>
</tr>
<tr>
<td>acl_resv_users</td>
<td>List of users who may and may not create reservations at this server.</td>
</tr>
<tr>
<td>resv_enable</td>
<td>Controls whether or not reservations can be created at this server.</td>
</tr>
</tbody>
</table>

The vnode attributes that affect reservations are listed here. See “Vnode Attributes” on page 366 of the PBS Professional Reference Guide for more information.

Table 4-15: Vnode Attributes Affecting Reservations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>Associates the vnode with an execution queue. If this attribute is set, this vnode cannot be used for reservations.</td>
</tr>
</tbody>
</table>
4.8.37.10  Reservation Advice and Caveats

- Do not delete a reservation’s queue.
- Do not start a reservation’s queue (do not set the reservation’s started attribute to True). Jobs will run prematurely.
- Do not use qmgr to set attribute values for a reservation queue.
- Reservations are incompatible with cycle harvesting. Do not allow reservations on machines used for cycle harvesting. The user may begin using the machine, which will suspend any PBS jobs, possibly preventing them from finishing before the reservation runs out. Set each cycle harvesting vnode’s resv_enable attribute to False, to prevent the vnode from being used for reservations.
- You can write hooks that execute, modifying a reservation’s attributes, when a reservation is created. See Chapter 6, "Hooks", on page 413.
- Allow enough time in reservations. If a job is submitted to a reservation with a duration close to the walltime of the job, provisioning could cause the job to be terminated before it finishes running, or to be prevented from starting. If a reservation is designed to take jobs requesting an AOE, leave enough extra time in the reservation for provisioning.
- The xpbs GUI cannot be used for creation, querying, or deletion of reservations.
- Hosts or vnodes that have been configured to accept jobs only from a specific queue (vnode-queue restrictions) cannot be used for advance reservations. Hosts or vnodes that are being used for cycle harvesting should not be used for reservations.
- Hosts with $max_load and $ideal_load configured should not be used for reservations.
Scheduling

Set the resv_enable vnode attribute on these hosts to False.

- For troubleshooting problems with reservations, see section 13.8.4, “Job in Reservation Fails to Run”, on page 905.
- Be careful when using qrun –H on jobs or vnodes involved in reservations. Make sure that you don’t oversubscribe reserved resources.
- In order to create reservations, the submission host must have its timezone set to a value that is understood by the PBS server. See section 13.6.14, “Unrecognized Timezone Variable”, on page 899.
- Avoid making reservations for resources that are out of the control of PBS. Resources that are managed through a server_dyn_res script may not be available when jobs need them.

4.8.38 Round Robin Queue Selection

PBS can select jobs from queues by examining the queues in round-robin fashion. When using the round-robin method, the scheduler considers the first queue, tries to run a job from that queue, then considers the next queue, tries to run a job from that queue, then considers the next queue, and so on, in a circular fashion.

You can specify whether PBS uses the round-robin method to select jobs, by setting the value of the round_robin scheduler attribute. When this attribute is True, PBS uses the round-robin method. See "round_robin" on page 294 of the PBS Professional Reference Guide.

The round_robin parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

The order in which queues are selected is determined by each queue’s priority, and whether you specify that queues should be sorted. You can set each queue’s priority; see section 2.2.5.3, “Prioritizing Execution Queues”, on page 23. You can specify that queues should be sorted by setting the sort_queues scheduler parameter to True. The sort_queues parameter is deprecated as of 12.2. See section 4.8.45, “Sorting Queues into Priority Order”, on page 278. If queue priorities are not set, they are undefined. If you do not sort the queues, their order is undefined.

You can use the round-robin method as a resource allocation tool. For example, if you need to run the same number of jobs from each group, you can put each group’s jobs in a different queue, and then use round-robin to run jobs, one from each queue.

The round-robin method is also used in PBS for some features that are not controlled by the round_robin scheduler attribute. They are the following:

- Routing queues try destinations in round-robin fashion, in the order listed
- The SMP cluster distribution parameter, smp_cluster_dist, can use a round-robin
Chapter 4  Scheduling

method to place jobs

4.8.38.1  Round-robin Caveats

- Each scheduling cycle starts with the highest-priority queue. Therefore, when using round-robin, this queue gets preferential treatment.
- When set to True, the round_robin parameter overrides the by_queue parameter.
- If round robin and strict ordering are True, and the top job cannot run, whether because of resources or rejection by MoM, no job runs. However, if round robin is True and strict ordering is False, and the top job in the current queue cannot run, the top job in the next queue is considered instead. For example, we have 3 queues, each with 3 jobs:

Q1: J1 J2 J3
Q2: J4 J5 J6
Q3: J7 J8 J9

If round_robin and strict_ordering are True, and J1 cannot run, no job runs.

If round_robin is True and strict_ordering is False, and J1 cannot run, job order is J4, J7, J2, J5, J8, J3, etc.
- With round_robin and strict_ordering set to True, a job continually rejected by a runjob hook may prevent other jobs from being run. A well-written hook would put the job on hold or requeue the job at some later time to allow other jobs in the same queue to be run.

4.8.39  Routing Jobs

Before reading this section, please read about the mechanics of configuring and using routing queues, in section 2.2.6, “Routing Queues”, on page 24.

In this section, we use the term “routing” to mean the general process of moving a job somewhere, whether it is from one queue to another, from one complex to another, or from a queue to particular vnodes.

Routing jobs can involve collecting jobs so they don’t stray into the wrong queues, moving those jobs to the correct queues, and filtering which jobs are allowed into queues.

You may need to collect jobs into a routing queue, before moving them to the correct destination queue. If you use a routing queue, you can force users to submit jobs to the routing queue only, you can grab jobs as they are submitted and put them in the routing queue, and you can set a routing queue as the default. The mechanisms to collect jobs are described below, and listed here:

- Setting default queue; see section 4.8.39.1.i, “Default Queue as Mechanism to Collect
Jobs”, on page 258

- Grabbing jobs upon submission; see section 4.8.39.1.ii, “Grabbing Jobs Upon Submission”, on page 258
- Disallowing direct submission to execution queues; see section 4.8.39.1.iii, “Disallowing Direct Submission as Mechanism to Collect Jobs”, on page 258
- Disallowing submission using access controls; see section 4.8.39.3.ii, “Access Controls as Filtering Mechanism”, on page 260

There is also a one-step process, but depending on the number of jobs being submitted, it may be too slow. You can also simply examine them upon submission and send them where you want. The method is listed here:

- Examining jobs upon submission and routing them using a hook; see section 4.8.39.1.iv, “Examining Jobs Upon Submission”, on page 258.

You can use any of several mechanisms for moving jobs. Each is described in subsections below. The mechanisms for moving jobs are the following:

- Routing Queues; see section 4.8.39.2.i, “Routing Queues as Mechanism to Move Jobs”, on page 259
- Hooks; see section 4.8.39.2.ii, “Hooks as Mechanism to Move Jobs”, on page 259
- Peer scheduling; see section 4.8.39.2.iii, “Peer Scheduling as Mechanism to Move Jobs”, on page 259
- The qmove command; see section 4.8.39.2.iv, “The qmove Command as Mechanism to Move Jobs”, on page 260

You can use filtering methods to control which jobs are allowed into destination queues. We describe filtering methods in subsections below. The filtering mechanisms are the following:

- Resource limits; jobs are filtered by resource request. See section 4.8.39.3.i, “Resource Limits as Filtering Mechanism”, on page 260
- Access control limits; jobs are filtered by owner. See section 4.8.39.3.ii, “Access Controls as Filtering Mechanism”, on page 260

You can use a combination of moving a job and “tagging” it, that is, including a special custom resource in the job’s resource request, to route the job. If you set the resource using a hook, you can route the job either to a queue or to vnodes. If you make the job inherit the resource from a queue, you can route it only to vnodes. You can set resource limits for the special custom resource at the receiving queue, allowing in only jobs with the special resource. You can set the special custom resource at vnodes, so that the job must run there. Mechanisms for tagging jobs are listed here:

- Using a hook to assign a resource; see section 4.8.39.4.i, “Using Hooks to Tag Jobs”, on
4.8.39.1 Mechanisms for Collecting Jobs

4.8.39.1.i Default Queue as Mechanism to Collect Jobs
To make it easy on your users, have their jobs land in your routing queue by default. You probably don’t want frustrated users trying to submit jobs without specifying a queue, only to have the jobs be rejected if you have set access controls on, or only allowed routing to, the default queue. The server’s default_queue attribute specifies the name of the default queue. To make things easy, make the default queue be the routing queue:

```
Qmgr: set server default_queue = <queue name>
```

4.8.39.1.ii Grabbing Jobs Upon Submission
You can allow users to submit jobs to any queue, and then scoop up the newly-submitted jobs and put them in the desired queue. To do this, you write a hook. There is a hook of this kind in the example "Redirecting newly-submitted jobs:" on page 422.

4.8.39.1.iii Disallowing Direct Submission as Mechanism to Collect Jobs
If you are using a routing queue, you can disallow job submission to all other queues. This forces users to submit jobs to the routing queue. You should probably make the routing queue be the default queue in this case, to avoid irritating users. Whether or not a queue allows direct job submission is controlled by its from_route_only attribute. To disallow job submission to a queue:

```
Qmgr: set queue <queue name> from_route_only = True
```

4.8.39.1.iv Examining Jobs Upon Submission
You can use a job submission hook to examine each job as it is submitted, and then route it to the desired queue. For example, you can route jobs directly according to resource request, project, owner, etc. See Chapter 6, "Hooks", on page 413.
4.8.39.2 Mechanisms for Moving Jobs

4.8.39.2.i Routing Queues as Mechanism to Move Jobs
Routing queues are a mechanism supplied by PBS that automatically move jobs from a routing queue to another queue. You can direct which destination queues accept a job using these filters at each destination queue:

• Resource limits: you can set up execution queues designed for specific kinds of jobs, and then route each kind of job separately. For example, you can create two execution queues, and one routing queue, and route all jobs requesting large amounts of memory to one of the execution queues, and the rest of the jobs to the other queue. See section 2.2.6.4, “Using Resources to Route Jobs Between Queues”, on page 25.

• Access control limits: you can set up destination queues that are designed for specific groups of users. Each queue accepts jobs only from a designated set of users or groups. For example, if you have three departments, Math, Physics, and Chemistry, the queue belonging to Math accepts only users from the Math department. See section 2.2.6.5, “Using Access Control to Route Jobs”, on page 30.

When routing a job between complexes, the job's owner must be able to submit a job to the destination complex.

For how to configure and use routing queues, see section 2.2.6, “Routing Queues”, on page 24.

4.8.39.2.ii Hooks as Mechanism to Move Jobs
You can use a submission hook to move jobs into queues such as dedicated time queues, queues with special priority, or reservation queues. You write the hook so that it identifies the jobs that should go into a particular queue, and then moves them there. For example, your hook can move all jobs from ProjectA to a specific queue. This is a snippet, where you would replace <destination queue> with the queue name.

```python
import pbs
e = pbs.event()
e.job.queue = pbs.server().queue("<destination queue>")
```

For complete information on hooks, see Chapter 6, “Hooks”, on page 413.

4.8.39.2.iii Peer Scheduling as Mechanism to Move Jobs
To send jobs from one complex to another, you use peer scheduling. In peer scheduling, the complex that supplies the jobs (the “furnishing” complex) contains at least one special queue (the “furnishing queue”), whose jobs can be pulled over to another complex, to be run at the other complex. The complex that pulls jobs contains a special queue (the “pulling queue”), where those pulled jobs land.
Chapter 4  

Scheduling

You can use any of the job routing methods, such as routing queues, tagging, or hooks, to control which jobs land in the furnishing queue.

You can use any of the job filtering methods, such as resource limits or access controls, to control which jobs land in the furnishing queue.

You can use job submission hooks on the jobs that land in the pulling queue.

See section 4.8.31, “Peer Scheduling”, on page 205.

4.8.39.2.iv  The qmove Command as Mechanism to Move Jobs

You can use the qmove command, either manually or via a cron job or the Windows Task Scheduler, to move jobs into the desired queues. See “qmove” on page 172 of the PBS Professional Reference Guide.

4.8.39.3  Mechanisms for Filtering Jobs

4.8.39.3.i  Resource Limits as Filtering Mechanism

You can filter whether each job is accepted at the server or a queue based on the job’s resource request. For example, you can control which jobs are allowed to be submitted to the server, by limiting the amount of memory a job is allowed to request. You can do the same at execution queues. These limits apply regardless of the routing mechanism being used, and apply to jobs being submitted directly to the queue. See section 5.13, “Using Resources to Restrict Server, Queue Access”, on page 318.

4.8.39.3.ii  Access Controls as Filtering Mechanism

You can filter jobs whether each job is accepted at the server or a queue based on the job’s owner, or the job owner’s group. At each queue and at the server, you can create a different list of the users who can submit jobs and the users who cannot submit jobs. You can do the same for groups.

For example, you can set up a routing queue and several execution queues, where each execution queue has access controls allowing only certain users and groups. When PBS routes the jobs from the routing queue, it will route them into the execution queues that accept owners of the jobs. See section 2.2.6.5, “Using Access Control to Route Jobs”; on page 30.

4.8.39.3.iii  Hooks as Filtering Mechanism

You can filter which jobs are accepted at the server or queues according to any criterion, using a hook. For example, you can write a hook that disallows jobs that request certain combinations of resources. See Chapter 6, “Hooks”; on page 413.
4.8.39.4 Mechanisms for Tagging Jobs

4.8.39.4.i Using Hooks to Tag Jobs

You can use a hook to force certain jobs to run on particular hardware, by having the hook set the value of a host-level custom resource in a job’s resource request. The hook sets this resource to match the value at the selected vnodes, so that the job must run on one or more of those vnodes. You can use the job’s project to determine how the job is tagged. Note that the value at other vnodes should be different, otherwise the job could end up on vnodes you don’t want.

- Define a host-level custom resource; see section 5.14.5, “Configuring Host-level Custom Resources”, on page 336.
- Set this resource to a special value on the special vnodes only. See section 5.7.2, “Setting Values for Global Static Resources”, on page 301.
- Create a hook that filters jobs by size, project, or other characteristic, and sets the value of the custom resource to the special value, in the job’s resource request. See Chapter 6, "Hooks", on page 413.

If you must use a routing queue, and you need to route on host-level resources (resources in the job’s select specification), you can use a hook to tag jobs so that they are routed correctly. The hook reads the job’s host-level resource request, and sets the job’s server-level resource request accordingly. This server-level resource is used for routing:

- Create a custom server-level resource that you use exclusively for routing; set it to appropriate values on the destination queues; see section 5.14.4, “Configuring Server-level Resources”, on page 334.
- Create a submit hook to extract the host-level resource value and use it to populate the custom resource that you use exclusively for routing; see Chapter 6, "Hooks", on page 413.

4.8.39.4.ii Using the qalter Command to Tag Jobs

You can change a job’s resource request using the qalter command. This way you can over-ride normal behavior. See “qalter” on page 127 of the PBS Professional Reference Guide.

4.8.40 Shared vs. Exclusive Use of Resources by Jobs

When PBS places a job, it can do so on hardware that is either already in use or has no jobs running on it. PBS can make the choice at the vnode level or at the host level. How this choice is made is controlled by a combination of the value of each vnode’s sharing attribute and the placement requested by a job.
You can set each vnode’s `sharing` attribute so that the vnode or host is always shared, always exclusive, or so that it honors the job’s placement request. The value of a vnode’s `sharing` attribute takes precedence over a job’s placement request.

Each vnode can be allocated exclusively to one job (each job gets its own vnodes), or its resources can be shared among jobs (PBS puts as many jobs as possible on a vnode). If a vnode is allocated exclusively to a job, all of its resources are assigned to the job. The state of the vnode becomes `job-exclusive`. No other job can use the vnode.

Hosts can also be allocated exclusively to one job, or shared among jobs.

For a complete description of the `sharing` attribute, and a table showing the interaction between the value of the `sharing` attribute and the job’s placement request, see "sharing" on page 371 of the PBS Professional Reference Guide.

### 4.8.40.0.i Sharing on a Shared-memory Altix

On a shared-memory Altix, the scheduler will share memory from a chunk even if all the CPUs are used by other jobs. It will first try to put a chunk entirely on one vnode. If it can, it will run it there. If not, it will break the chunk up across any vnode it can get resources from, even for small amounts of unused memory.

### 4.8.40.1 Setting the sharing Vnode Attribute

When setting the `sharing` vnode attribute, follow the rules in section 3.5.2, “Choosing Configuration Method”, on page 52.

### 4.8.40.2 Viewing Sharing Information

You can use the `qmgr` or `pbsnodes` commands to view sharing information. See “`qmgr`” on page 150 of the PBS Professional Reference Guide and “`pbsnodes`” on page 103 of the PBS Professional Reference Guide.

### 4.8.40.3 Sharing Caveats

- On the Cray, the `sharing` attribute is set to `force_exclhost` by default. Do not change this setting.
- The term “sharing” is also used to describe the case where MoM manages a resource that is shared among her vnodes, for example a license shared by the vnodes of a multi-vnode machine.
- The term “sharing” is also used to mean oversubscribing CPUs, where more than one job is run on one CPU; the jobs are “sharing” a CPU. See section 9.4.5, “Managing Load.”
Levels on Vnodes”, on page 738

• If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either `default_exclhost` or `force_exclhost`, all vnodes on that host must have the same value for the sharing attribute. When the MoM starts or restarts, if any vnode on a host is set to either `default_exclhost` or `force_exclhost`, and another vnode is set to a different value, the MoM will exit and log the following error message at event class 0x0001:

It is erroneous to mix sharing= <sharing val> for vnode <name> with sharing= <force_exclhost|default_exclhost> which is set for other vnodes on host <host>

• For vnodes with sharing=default_shared, jobs can share a vnode, so that unused memory on partially-allocated vnodes is allocated to a job. The `exec_vnode` attribute will show this allocation.

4.8.41 Using Shrink-to-fit Jobs

4.8.41.1 Shrink-to-fit Jobs

PBS allows you or the job submitter to adjust the running time of a job to fit into an available scheduling slot. The job’s minimum and maximum running time are specified in the `min_walltime` and `max_walltime` resources. PBS chooses the actual walltime. Any job that requests `min_walltime` is a shrink-to-fit job.

4.8.41.1.i Requirements for a Shrink-to-fit Job

A job must have a value for `min_walltime` to be a shrink-to-fit job. Shrink-to-fit jobs are not required to request `max_walltime`, but it is an error to request `max_walltime` and not `min_walltime`.

Jobs that do not have values for `min_walltime` are not shrink-to-fit jobs, and their walltime can be specified by the user, inherited through defaults, or set in a hook.

4.8.41.1.ii Comparison Between Shrink-to-fit and Non-shrink-to-fit Jobs

Shrink-to-fit jobs are treated the same as non-shrink-to-fit jobs unless explicitly stated. For example, job priority is not affected by being shrink-to-fit. The only difference between a shrink-to-fit and a non-shrink-to-fit job is how the job’s walltime is treated. PBS sets the walltime at the time the job is run; any walltime settings not computed by PBS are ignored.
4.8.41.2 Where to Use Shrink-to-fit Jobs

If you have jobs that can run for less than the expected time to completion and still make useful progress, you can use them as shrink-to-fit jobs in order to maximize utilization.

You can use shrink-to-fit jobs for the following:

- Jobs that are internally checkpointed. This includes jobs which are part of a larger effort, where a job does as much work as it can before it is killed, and the next job in that effort takes up where the previous job left off.
- Jobs using periodic PBS checkpointing
- Jobs whose real running time might be much less than the expected time
- When you have set up dedicated time for system maintenance, and you want to keep machines well-utilized right up until shutdown, submitters who want to risk having a job killed before it finishes can run speculative shrink-to-fit jobs. Similarly, speculative jobs can take advantage of the time just before a reservation starts
- Any job where the submitter does not mind running the job as a speculative attempt to finish some work

4.8.41.3 Running Time of a Shrink-to-fit Job

4.8.41.3.i Setting Running Time Range for Shrink-to-fit Jobs

It is only required that the job request min_walltime to be a shrink-to-fit job. If a job requests min_walltime but does not request max_walltime, you may want to use a hook or defaults to set a reasonable value for max_walltime. If you use defaults, you may want to route shrink-to-fit jobs to a special queue where they inherit a value for max_walltime if they haven’t got one already. See section 4.8.39, “Routing Jobs”, on page 256.

Requesting max_walltime without requesting min_walltime is an error.

A job can end up with a value for min_walltime and max_walltime when the user specifies them, when it inherits them from server or queue defaults, or when they are set in a hook.

Job submitters can set the job’s running time range by requesting min_walltime and max_walltime, for example:

```
qsub -l min_walltime=<min walltime>, max_walltime=<max walltime> <job script>
```

You can set min_walltime or max_walltime using a hook, whether or not the job requests it. You can set up defaults so that the job inherits these resources if they are not explicitly requested or set in a hook.
4.8.41.3.ii  Inheriting Values for min_walltime and max_walltime

The min_walltime and max_walltime resources inherit values differently. A job can inherit a value for max_walltime from resources_max.walltime; the same is not true for min_walltime. This is because once a job is shrink-to-fit, PBS can use a walltime limit for max_walltime.

If a job is submitted without a value for min_walltime, the value for min_walltime for the job becomes the first of the following that exists:

- Server’s default qsub arguments
- Queue’s resources_default.min_walltime
- Server’s resources_default.min_walltime

If a shrink-to-fit job is submitted without a value for max_walltime, the value for max_walltime for the job becomes the first of the following that exists:

- Server’s default qsub arguments
- Queue’s resources_default.max_walltime
- Server’s resources_default.max_walltime
- Queue’s resources_max.walltime
- Server’s resources_max.walltime

4.8.41.3.iii  Setting walltime for Shrink-to-fit Jobs

For a shrink-to-fit job, PBS sets the walltime resource based on the values of min_walltime and max_walltime, regardless of whether walltime is specified for the job. You cannot use a hook to set the job’s walltime, and any queue or server defaults for walltime are ignored, except for the case where the job is run via qrun -H; see section 4.8.41.8.ii, “Using qrun With -H Option”, on page 268.

PBS examines each shrink-to-fit job when it gets to it, and looks for a time slot whose length is between the job’s min_walltime and max_walltime. If the job can fit somewhere, PBS sets the job’s walltime to a duration that fits the time slot, and runs the job. The chosen value for walltime is visible in the job’s Resource_List.walltime attribute. Any existing walltime value, regardless of where it comes from (user, queue default, hook, previous execution), is reset to the new calculated running time.

If a shrink-to-fit job is run more than once, PBS recalculates the job’s running time to fit an available time slot that is between min_walltime and max_walltime, and resets the job’s walltime, each time the job is run.
4.8.41.4 How PBS Places Shrink-to-fit Jobs

The PBS scheduler treats shrink-to-fit jobs the same way as it treats non-shrink-to-fit jobs when it schedules them to run. The scheduler looks at each job in order of priority, and tries to run it on available resources. If a shrink-to-fit job can be shrunk to fit in an available slot, the scheduler runs it in its turn. The scheduler chooses a time slot that is at least as long as the job’s min_walltime value. A shrink-to-fit job may be placed in a time slot that is shorter than its max_walltime value, even if a longer time slot is available.

For a multi-vnode job, PBS chooses a walltime that works for all of the chunks required by the job, and places job chunks according to the placement specification.

4.8.41.5 Shrink-to-fit Jobs and Time Boundaries

The time boundaries that constrain job running time are the following:

- Reservations
- Dedicated time
- Primetime
- Start time for a top job

Time boundaries are not affected by shrink-to-fit jobs.

A shrink-to-fit job can shrink to avoid time boundaries, as long as the available time slot before the time boundary is greater than min_walltime.

If any job is already running, whether or not it is shrink-to-fit, and you introduce a new period of dedicated time that would impinge on the job’s running time, PBS does not kill or otherwise take any action to prevent the job from hitting the new boundary.

4.8.41.5.i Shrink-to-fit Jobs and Prime Time

If you have enabled prime time by setting backfill_prime to True, shrink-to-fit jobs will honor the boundary between primetime and non-primetime. If prime_spill is True, shrink-to-fit jobs are scheduled so that they cross the prime-nonprime boundary by up to prime_spill duration only. If prime_exempt_anytime_queues is set to True, a job submitted in an anytime queue is not affected by primetime boundaries.
4.8.41.6 Shrink-to-fit Jobs and Resource Limits

4.8.41.6.i Shrink-to-fit Jobs and Gating at Server or Queue
Shrink-to-fit jobs must honor any resource limits at the server or queues. If a walltime limit is specified:

- Both min_walltime and max_walltime must be greater than or equal to resources_min.walltime.
- Both min_walltime and max_walltime must be less than or equal to resources_max.walltime.

If resource limits are not met, a job submission or modification request will fail with the following error:

“Job exceeds queue and/or server resource limits”

4.8.41.6.ii Gating Restrictions
You cannot set resources_min or resources_max for min_walltime or max_walltime. If you try, you will see the following error message, for example for min_walltime:

“Resource limits can not be set for min_walltime”

4.8.41.7 Shrink-to-fit Jobs and Preemption
When preempting other jobs, shrink-to-fit jobs do not shrink. Their walltime is set to their max_walltime.

4.8.41.8 Using qrun on Shrink-to-fit Jobs
If you use qrun on a shrink-to-fit job, its behavior depends on whether you use the -H option to qrun.

4.8.41.8.i Using qrun Without -H Option
When a shrink-to-fit job is run via qrun, it can shrink into available space to run. However, if preemption is enabled and there is a preemptable job that must be preempted in order to run the shrink-to-fit job, the preemptable job is preempted and the shrink-to-fit job shrinks and runs.

When a shrink-to-fit job is run via qrun, and there is a hard deadline, e.g. reservation or dedicated time, that conflicts with the shrink-to-fit job’s max_walltime but not its min_walltime, the following happens:

- If preemption is enabled and there is a preemptable job before the hard deadline that must be preempted in order to run the shrink-to-fit job, preemption behavior means that the
shrink-to-fit job does not shrink to fit; instead, it conflicts with the deadline and does not run.

- If preemption is enabled and there is no preemptable job before the hard deadline, the shrink-to-fit job shrinks into the available time and runs.

4.8.41.8.ii  Using `qrun` With `-H` Option

When a shrink-to-fit job is run via `qrun -H`, the shrink-to-fit job runs, regardless of reservations, dedicated time, other jobs, etc. When run via `qrun -H`, shrink-to-fit jobs do not shrink. If the shrink-to-fit job has a requested or inherited value for `walltime`, that value is used, instead of one set by PBS when the job runs. If no `walltime` is specified, the job runs without a `walltime`.

4.8.41.9  Modifying Shrink-to-fit and Non-shrink-to-fit Jobs

4.8.41.9.i  Modifying `min_walltime` and `max_walltime`

You can change `min_walltime` and/or `max_walltime` for a shrink-to-fit job using `modifyjob` or `queuejob` hooks, or by using the `qalter` command. Any changes take effect after the current scheduling cycle. Changes affect only queued jobs; running jobs are unaffected unless they are rerun.

4.8.41.9.ii  Making Non-shrink-to-fit Jobs into Shrink-to-fit Jobs

You can convert a normal non-shrink-to-fit job into a shrink-to-fit job using the following methods:

- Use a hook that does the following:
  - Sets `max_walltime` to the job’s `walltime`
  - Sets `min_walltime` to a useful value
- Use `resources_default` at the server or a queue. For a queue, you might want to set that queue’s `from_route_only` attribute to True.
- Route to a queue that has `resources_default.min_walltime` set.
- Use the `qalter` command to set values for `min_walltime` and `max_walltime`.

Any changes take effect after the current scheduling cycle. Changes affect only queued jobs; running jobs are unaffected unless they are rerun.

4.8.41.9.iii  Making Shrink-to-fit Jobs into Non-shrink-to-fit Jobs

To make a shrink-to-fit job into a normal, non-shrink-to-fit job, use either a hook or the `qalter` command to do the following:

- Set the job’s `walltime` to the value for `max_walltime` (beware of allowing the job to run
into existing reservations etc.)

- Unset min.walltime
- Unset max.walltime

4.8.41.9.iv   Hooks for Running Time Limits

If you want to set a new running time limit for shrink-to-fit jobs, you can use a hook. However, this hook must set the value of max.walltime, rather than walltime, since hook settings for walltime for a shrink-to-fit job are ignored.

4.8.41.10   Viewing Running Time for a Shrink-to-fit Job

4.8.41.10.i   Viewing min.walltime and max.walltime

You can use qstat -f to view the values of the min.walltime and max.walltime. For example:

```
% qsub -lmin_walltime=01:00:15, max_walltime=03:30:00 job.sh
<job-id>
% qstat -f <job-id>
...
resource_list.min_walltime=01:00:15
resource_list.max_walltime=03:30:00
```

You can use tracejob to display max.walltime and min.walltime as part of the job’s resource list. For example:

```
12/16/2011 14:28:55  A    user=pbsadmin group=Users
                      project=_pbs_project_default
...
Resource_List.max_walltime=10:00:00
Resource_List.min_walltime=00:00:10
```

4.8.41.10.ii   Viewing walltime for a Shrink-to-fit Job

PBS sets a job’s walltime only when the job runs. While the job is running, you can see its walltime via qstat -f. While the job is not running, you cannot see its real walltime; it may have a value set for walltime, but this value is ignored.

You can see the walltime value for a finished shrink-to-fit job if you are preserving job history. See section 11.15, “Managing Job History”, on page 852.

You can see the walltime value for a finished shrink-to-fit job in the scheduler log.
4.8.41.11 Lifecycle of a Shrink-to-fit Job

4.8.41.11.i Execution of Shrink-to-fit Jobs
Shrink-to-fit jobs are started just like non-shrink-to-fit jobs.

4.8.41.11.ii Termination of Shrink-to-fit Jobs
When a shrink-to-fit job exceeds the walltime PBS has set for it, it is killed by PBS exactly as a non-shrink-to-fit job is killed when it exceeds its walltime.

4.8.41.12 The min_walltime and max_walltime Resources

max_walltime
Maximum walltime allowed for a shrink-to-fit job. Job’s actual walltime is between max_walltime and min_walltime. PBS sets walltime for a shrink-to-fit job. If this resource is specified, min_walltime must also be specified. Must be greater than or equal to min_walltime. Cannot be used for resources_min or resources_max. Cannot be set on job arrays or reservations. If not specified, PBS uses an eternal time slot. Can be requested only outside of a select statement. Non-consumable. Default: None. Type: duration. Python type: pbs.duration

min_walltime
Minimum walltime allowed for a shrink-to-fit job. When this resource is specified, job is a shrink-to-fit job. If this attribute is set, PBS sets the job’s walltime. Job’s actual walltime is between max_walltime and min_walltime. Must be less than or equal to max_walltime. Cannot be used for resources_min or resources_max. Cannot be set on job arrays or reservations. Can be requested only outside of a select statement. Non-consumable. Default: None. Type: duration. Python type: pbs.duration

4.8.41.13 Accounting and Logging for Shrink-to-fit Jobs

4.8.41.13.i Accounting Log Entries for min_walltime and max_walltime
The accounting log will contain values for min_walltime and max_walltime, as part of the job’s Resource_List attribute. This attribute is recorded in the S, E, and R records in the accounting log. For example, if the following job is submitted:

qsub -l min_walltime="00:01:00",max_walltime="05:00:00" -l select=2:ncpus=1 job.sh

PBS Professional 12.2 Administrator’s Guide
This is the resulting accounting record:

```
...S...... Resource_List.max_walltime=05:00:00
        Resource_List.min_walltime=00:01:00 Resource_List.ncpus=2
        Resource_List.nodeect=2 Resource_List.place=pack
        Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18
        resource_assigned.ncpus=2

...R...... Resource_List.max_walltime=05:00:00
        Resource_List.min_walltime=00:01:00 Resource_List.ncpus=2
        Resource_List.nodeect=2 Resource_List.place=pack
        Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18

...E...... Resource_List.max_walltime=05:00:00
        Resource_List.min_walltime=00:01:00 Resource_List.ncpus=2
        Resource_List.nodeect=2 Resource_List.place=pack
        Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18......
```

### 4.8.41.13.ii Logging

- When the scheduler finds a primetime/dedicated time conflict with a shrink-to-fit job, and the job can be shrunk, the following message is logged in the scheduler logs, with log level PBSEVENT_DEBUG2:

  “Considering shrinking job to duration=<duration>, due to prime/dedicated time conflict”

  Sample message from the scheduler log:

  “03/26/2012 11:53:55;0040;pbs_sched;Job;98.blrlap203;Considering shrinking job to duration=1:06:05, due to a prime/dedicated time conflict”

  This message doesn't indicate or guarantee that the job will eventually be shrunk and run. This message shows that the job's maximum running time conflicted with primetime and the job can still be run by shrinking its running time.

- When the scheduler finds a reservation/top job conflict with a shrink-to-fit job, and the job can be shrunk, the following message is logged in the scheduler logs, with log level
PBSEVENT_DEBUG2:
“Considering shrinking job to duration=<duration>”, due to reservation/top job conflict"

Sample log message from the scheduler log:
“03/26/2012 11:53:55;0040;pbs_sched;Job;98.blrlap203; Considering shrinking job to duration=1:06:05, due to reservation/top job conflict”

This message doesn’t indicate or guarantee that the job will eventually be shrunk and run. This message shows that the job’s maximum running time conflicted with a reservation or top job and the job can still be run by shrinking its running time.

• When the scheduler runs the shrink-to-fit job, the following message is logged in the scheduler logs with log level PBSEVENT_DEBUG2:
“Job will run for duration=<duration>”

Sample scheduler log message:
“03/26/2012 11:53:55;0040;pbs_sched;Job;98.blrlap203;Job will run for duration=1:06:05”

4.8.41.14 Caveats and Restrictions for Shrink-to-fit Jobs

• It is erroneous to specify max_walltime for a job without specifying min_walltime. If a queuejob or modifyjob hook attempts this, the following error appears in the server logs. If attempted via qsub or qalter, the following error appears in the server log and is printed as well:
’Can not have “max_walltime” without “min_walltime”’

• It is erroneous to specify a min_walltime that is greater than max_walltime. If a queuejob or modifyjob hook attempts this, the following error appears in the server logs. If attempted via qsub or qalter, the following error appears in the server log and is printed as well:
’“min_walltime” can not be greater than “max_walltime”’

• Job arrays cannot be shrink-to-fit. You cannot have a shrink-to-fit job array. It is erroneous to specify a min_walltime or max_walltime for a job array. If a queuejob or modifyjob hook attempts this, the following error appears in the server logs. If attempted via qsub or qalter, the following error appears in the server log and is printed as well:
’“min_walltime” and “max_walltime” are not valid resources for a job array’

• Reservations cannot be shrink-to-fit. You cannot have a shrink-to-fit reservation. It is erroneous to set min_walltime or max_walltime for a reservation. If attempted via
pbs_rsub, the following error is printed:
'"min_walltime" and "max_walltime" are not valid resources for reservation.'
• It is erroneous to set resources_max or resources_min for min_walltime and max_walltime. If attempted, the following error message is displayed, whichever is appropriate:
  “Resource limits can not be set for min_walltime”
  “Resource limits can not be set for max_walltime”

4.8.42 SMP Cluster Distribution

PBS provides a method for distributing single-chunk jobs to a cluster of single-vnode machines according to a simple set of rules. The method is called SMP cluster distribution. It takes into account the resources specified on the resources: line in PBS_HOME/sched_priv/sched_config. The SMP cluster distribution method allows you to choose one of three job distribution systems:

Table 4-16: SMP Cluster Distribution Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>pack</td>
<td>Pack all jobs onto one vnode, until that vnode is full, then move to the next vnode</td>
</tr>
<tr>
<td>round_robin</td>
<td>Place one job on each vnode in turn, before cycling back to the first vnode</td>
</tr>
<tr>
<td>lowest_load</td>
<td>Place the job on the host with the lowest load average</td>
</tr>
</tbody>
</table>

4.8.42.1 How to Use SMP Cluster Distribution

To use SMP cluster distribution, do the following:
• Set the smp_cluster_dist scheduler parameter to the desired value. For example, to enable SMP cluster distribution using the round robin algorithm during primetime, and the pack algorithm during non-primetime, set the following in the scheduler’s configuration file:
  smp_cluster_dist: round_robin prime
  smp_cluster_dist: pack non_prime
• Set resources_available.<resource> to the desired limit on each vnode. You do not need to set any of the resources that are automatically set by PBS. For a list of these, see
Specify the resources to use during scheduling, in PBS_HOME/sched_priv/sched_config:

resources: "ncpus, mem, arch, host, ..."

The smp_cluster_dist parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

4.8.42.2 How To Disable SMP Cluster Distribution

To ensure that SMP cluster distribution does not interfere with your scheduling policy, leave the smp_cluster_dist parameter set to its default value:

smp_cluster_dist pack all

4.8.42.3 SMP Cluster Distribution Caveats and Advice

- This feature was intended for early implementations of complexes, and probably is not useful for you.
- If you use this feature, you are committed to using it for the entire complex; you cannot designate some machines where it will be used and others where it will not be used.
- If smp_cluster_dist with either round_robin or lowest_load is used with node_sort_key set to unused or assigned, smp_cluster_dist is set to pack.
- The avoid_provision provisioning policy is incompatible with the smp_cluster_dist scheduler configuration parameter. If a job requests an AOE, the avoid_provision policy overrides the behavior of smp_cluster_dist.
- This feature is applied only to single-chunk jobs that specify an arrangement of pack. Multi-chunk jobs are ignored.
- This feature is useful only for single-vnode machines. On a multi-vnoded machine, this feature distributes jobs across vnodes, but those jobs can end up all stuck on a single host.
- The choice of smp_cluster_dist with round_robin can be replaced by sorting vnodes according to unused CPUs, which does a better job:

node_sort_key: “ncpus HIGH unused”
4.8.43 Sorting Jobs on a Key

PBS allows you to sort jobs on a key that you specify. This can be used when setting both execution and preemption priority. Sorting jobs comes into play after jobs have been divided into classes, because each class may contain more than one job. You can sort on one or more of several different keys, and for each key, you can sort either from low to high or from high to low.

You configure sorting jobs on a key by setting values for the job_sort_key scheduler parameter.

You can create an invisible, unrequestable custom resource, and use a hook to set the value of this resource for each job. The hook would modify the job’s resource request to include the new resource, and set the value to whatever the hook computes. Then you can sort jobs according to the value of this resource.

The job_sort_key parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

4.8.43.1 job_sort_key Syntax

job_sort_key: “<sort key> HIGH | LOW <primetime option>”

You can use the following keys for sorting jobs:

<table>
<thead>
<tr>
<th>Sort Key</th>
<th>Allowed Order</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;PBS resource&gt;</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>fair_share_perc</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>job_priority</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
</tbody>
</table>
You can sort on up to 20 keys.

The argument to the job_sort_key parameter is a quoted string. The default for job_sort_key is that it is not in force.

See “job_sort_key” on page 285 of the PBS Professional Reference Guide.

### 4.8.43.2 Configuring Sorting Jobs on a Key

You can specify more than one sort key, where you want a primary sort key, a secondary sort key, etc.

If you specify more than one entry for job_sort_key, the first entry is the primary sort key, the second entry is the secondary sort key, which is used to sort equal-valued entries from the first sort, and so on.

Each entry is specified one to a line.

To sort jobs on a key, set the job_sort_key scheduler parameter:

- Set the desired key
- Specify whether high or low results should come first
- Specify the primetime behavior

The scheduler’s configuration file is read on startup and HUP.

<table>
<thead>
<tr>
<th>Sort Key</th>
<th>Allowed Order</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preempt_priority</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>sort_priority</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
</tbody>
</table>
4.8.43.3 Examples of Sorting Jobs on Key

Example 4-26: Sort jobs so that those with long walltime come first:

```sh
job_sort_key: "walltime HIGH"
```

Example 4-27: For example, if you want big jobs to run first, where “big” means more CPUs, and if the CPUs are the same, more memory, sort on the number of CPUs requested, then the amount of memory requested:

```sh
job_sort_key: "ncpus HIGH" all
job_sort_key: "mem HIGH" all
```

Example 4-28: Sort jobs so that those with lower memory come first:

```sh
job_sort_key: "mem LOW" prime
```

Example 4-29: Sort jobs so that those with highest preemption priority come first:

```sh
job_sort_key: "preempt_priority HIGH"
```

Example 4-30: Sort jobs according to the value of an invisible custom resource called JobOrder:

```sh
job_sort_key: "JobOrder LOW" all
```

4.8.43.4 Caveats and Advice for Sorting Jobs on Key

- Do not use `fair_share_perc` as the sort key when using fairshare, meaning the `fair_share` scheduler parameter is enabled. If you do this, the scheduler will attempt to sort a set of jobs where each job has the same sort key value. This will not sort the jobs.

- Use the `fair_share_perc` option only when ordering jobs by entity shares. See section 4.8.14, “Sorting Jobs by Entity Shares (Was Strict Priority)”, on page 160.

- If you are sorting jobs using `job_sort_key`, we recommend that when you change `preempt_prio` from the default, you add `preempt_priority` as the primary key for the `job_sort_key` scheduler parameter. This is especially important when using soft limits for preemption. Jobs that are over their soft limits should have lower priority, both in execution and preemption. For example, if you’re using soft limit preemption levels, but not assigning job execution priority according to soft limits, you can have job A started and preempted in the same scheduler cycle, because job A has higher execution priority, but is over its soft limits, and has lower preemption priority. To sort jobs on preemption...
Chapter 4  Scheduling

priority:
job_sort_key: “preempt_priority HIGH”

• To run big jobs first, use ncpus as the primary sort key for job_sort_key:
  job_sort_key: “ncpus HIGH”

• The job_sort_key parameter is overridden by the job sorting formula and by fairshare. If it is invalid to set both job_sort_formula and job_sort_key at the same time. If they are both set, job_sort_key is ignored and the following error message is logged:
  “Job sorting formula and job_sort_key are incompatible. The job sorting formula will be used.”

• The scheduler’s configuration file contains an example line for job_sort_key. This line is commented out, but shows an example of job_sort_key with “cput” as the sorting key.

4.8.44 Sorting Jobs by Requested Priority

You can sort jobs according to the priority that was requested for the job. This value is found in the job’s Priority attribute. You can use this value in the following ways:

• The term job_priority represents the value of the job’s priority attribute in the job sorting formula. See section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.

• The job_sort_key scheduler parameter can take the term job_priority as an argument. The term job_priority represents the value of the job’s Priority attribute. See section 4.8.43, “Sorting Jobs on a Key”, on page 275.

You can use a hook to set or change the value of a job’s Priority attribute. See section “Hooks”, on page 413.

4.8.45 Sorting Queues into Priority Order

You can tell PBS to sort all the execution queues in your complex according to their priority, and then to use that ordering when examining queues individually. Using this, queues are ordered with the highest-priority queue first.

If you want queues to be considered in a specific order, you must assign a different priority to each queue, and set the sort_queues scheduler parameter to True. When the sort_queues scheduler parameter is True, the scheduler sorts the queues according to their priority.

To do this, give the queue you want considered first the highest priority, then the next queue the next highest priority, and so on. To set a queue’s priority, use the qmgr command to assign a value to the priority queue attribute.

Qmgr: set queue <queue name> priority = <value>
The `sort_queues` parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time. See “`sort_queues`” on page 295 of the PBS Professional Reference Guide.

Sorting queues into priority order is useful for the following:

- Examining queues one at a time. See section 4.8.4, “Examining Jobs Queue by Queue”, on page 129.

### 4.8.45.1 Caveats and Advice when Sorting Queues

- If you do not set queue priorities, their ordering is undefined.
- The `sort_queues` parameter is useful only when PBS is looking at individual queues, such as when using `by_queue` or `round_robin`. If all jobs in the complex are being considered as if they were in one queue, `sort_queues` has no effect.
- The `sort_queues` parameter is deprecated as of 12.2; it will not be necessary in later releases.

### 4.8.46 Starving Jobs

PBS can keep track of the amount of time a job has been waiting to run, and then mark the job as _starving_ if this time has passed a specified limit. You can use this starving status in calculating both execution and preemption priority.

#### 4.8.46.1 Enabling Starving

You enable tracking whether jobs are starving by setting the `help_starving_jobs` scheduler parameter to `True`.

You specify the amount of time required for a job to be considered starving in the `max_starve` scheduler parameter. The default for this parameter is 24 hours.

The `help_starving_jobs` parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time. See “`help_starving_jobs`” on page 284 of the PBS Professional Reference Guide.

#### 4.8.46.2 Time Used for Starving

PBS can use one of the following kinds of time to determine whether a job is starving:

- The job’s eligible wait time, described in section 4.8.13, “Eligible Wait Time for Jobs”.
The amount of time the job has been queued

You specify which to use in the server’s `eligible_time_enable` attribute. When `eligible_time_enable` is set to `True`, each job’s `eligible_time` value is used as its wait time for starving. If `eligible_time_enable` is set to `False`, the amount of time the job has been queued is used as its wait time for starving. The default for `eligible_time_enable` is `False`.

If the server’s `eligible_time_enable` attribute is set to `False`, the following rules apply:

- The amount of time the job has been queued is used as its wait time for starving.
- Jobs lose their queue wait time whenever they are requeued, as with the `qrerun` command. This includes when they are checkpointed or requeued (but not suspended) during preemption.
- Suspended jobs do not lose their queue wait time. However, when they become suspended, the amount of time since they were submitted is counted towards their queue wait time. For example, if a job was submitted, then remained queued for 1 hour, then ran for 26 hours, then was suspended, if `max_starve` is 24 hours, then the job will become starving.

If the server’s `eligible_time_enable` attribute is set to `True`, the following rules apply:

- The job’s `eligible_time` value is used as its wait time for starving.
- Jobs do not lose their `eligible_time` when they are requeued.
- Jobs do not lose their `eligible_time` when they are suspended.

4.8.46.3 Starving and Job Priority

Starving is one of the job classes used by PBS to calculate job execution priority. If you enable starving jobs, PBS will classify starving jobs in the Starving class, which gives them greater than ordinary priority. See section 4.8.16, “Calculating Job Execution Priority”, on page 166. Each job’s eligible wait time can also be used in the job sorting formula used to calculate job execution priority. See section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.

Starving is one of the job classes that you can use when specifying how preemption should work. You can choose how much preemption priority is given to starving jobs when you set preemption levels. See section 4.8.33, “Using Preemption”, on page 228.
4.8.46.4 Parameters and Attributes Affecting Starving

The following table lists the parameters and attributes that affect starving:

Table 4-18: Parameters and Attributes Affecting Starving

<table>
<thead>
<tr>
<th>Parameter or Attribute</th>
<th>Location</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>help_starving_jobs</td>
<td>PBS_HOME/sched_priv/sched_config</td>
<td>Controls whether long-waiting jobs are considered starving. When set to True, jobs can be starving. Default: True all</td>
</tr>
<tr>
<td>max_starve</td>
<td>PBS_HOME/sched_priv/sched_config</td>
<td>Amount of wait time for job to be considered starving. Default: 24 hours.</td>
</tr>
<tr>
<td>eligible_time_enable</td>
<td>Server attribute</td>
<td>Controls whether a job’s wait time is taken from its eligible_time or from its queued time. When set to True, a job’s eligible_time is used as its wait time. Default: False.</td>
</tr>
<tr>
<td>eligible_time</td>
<td>Job attribute</td>
<td>The amount of time a job has been blocked from running due to lack of resources.</td>
</tr>
</tbody>
</table>

4.8.46.5 Starving and Queued or Running Jobs

When a job is running, it keeps the starving status it had when it was started. While a job is running, if it wasn’t starving before, it can’t become starving. However, it keeps its starving status if it became starving while queued.

4.8.46.6 Starving and Subjobs

Subjobs that are queued can become starving. Starving status is applied to individual subjobs in the same way it is applied to jobs. The queued subjobs of a job array can become starving while others are running. If a job array has starving subjobs, then the job array is starving.
4.8.46.7 Starving and Backfilling

Because a starving job can become a top job, but can continue to be unable to run due to a lack of resources, you may find it useful to use backfilling around starving jobs. See section 4.8.3, “Using Backfilling”, on page 124.

4.8.46.8 Starving Caveats

Do not enable starving with fairshare, meaning do not set both the fair_share and help_starving_jobs scheduler parameters to True.

4.8.47 Using Strict Ordering

By default, when scheduling jobs, PBS orders jobs according to execution priority, then considers each job, highest-priority first, and runs the next job that can run now. Using strict ordering means that you tell PBS that it must not skip a job when choosing which job to run. If the top job cannot run, no job runs.

Strict ordering does not change how execution priority is calculated.

4.8.47.1 Configuring Strict Ordering

To configure strict ordering, set the strict_ordering scheduler parameter to True.

The strict_ordering parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time. See “strict_ordering” on page 295 of the PBS Professional Reference Guide.

4.8.47.2 How Strict Ordering Works

Strict ordering does not affect how jobs are prioritized. When strict_ordering is True, the scheduler runs jobs in exactly the order of their priority. If jobs are being run queue by queue, that order is honored.

4.8.47.3 Combining Strict Ordering and Backfilling

Strict ordering alone may cause some resources to stand idle while the top job waits for resources to become available. If you want to prevent this, you can use backfilling with strict ordering. Using backfilling, if the top job cannot run, filler jobs can be squeezed in around the job that cannot run. See section 4.8.3, “Using Backfilling”, on page 124.
4.8.47.4 Strict Ordering Caveats

- It is inadvisable to use strict ordering and backfilling with fairshare. The results may be non-intuitive. Fairshare will cause relative job priorities to change with each scheduling cycle. It is possible that a job from the same entity or group will be chosen as the filler job. The usage from these filler jobs will lower the priority of the top job.

For example, if a user has a large job that is the top job, and that job cannot run, smaller jobs owned by that user will chew up the user's usage, and prevent the large job from being likely to ever run. Also, if the small jobs are owned by a user in one area of the fairshare tree, no large jobs owned by anyone else in that section of the fairshare tree are likely to be able to run.

- Using dynamic resources with strict ordering and backfilling may result in unpredictable scheduling. See section 4.8.3.8, “Backfilling Recommendations and Caveats”, on page 127.

- Using preemption with strict ordering and backfilling may change which job is being backfilled around.

- With both round robin and strict ordering, a job continually rejected by a runjob hook may prevent other jobs from being run. A well-written hook would put the job on hold or requeue the job at some later time to allow other jobs in the same queue to be run.

4.8.48 Sorting Vnodes on a Key

PBS can sort vnodes according to a key that you specify. This can be used when deciding which vnodes to use for jobs. Sorting vnodes comes into play after a placement set has been selected, or when a job will run on vnodes associated with a queue, or when placement sets are not used, because in those cases there may be more vnodes available than are needed. You can sort vnodes on one or more different keys, and for each key, you can sort from high to low, or the reverse. The default way to sort vnodes is according to the value of the vnode priority attribute, from higher to lower.

When you sort vnodes according to the assigned or unused amount of a resource, the vnode list is re-sorted after every job is run. This is because each job may change the usage for that resource.

You configure sorting vnodes on a key by setting values for the node_sort_key scheduler parameter.

The node_sort_key parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

When vnodes are not sorted on a key, their order is undefined.
4.8.48.1 node_sort_key Syntax

node_sort_key: “sort_priority HIGH | LOW” <prime option>
node_sort_key: “<resource> HIGH | LOW” <prime option>
node_sort_key: “<resource> HIGH | LOW total | assigned | unused” <prime option>

where

total
  Use the resources_available value
assigned
  Use the resources_assigned value
unused
  Use the value given by resources_available - resources_assigned

Specifying a resource such as mem or ncpus sorts vnodes by the resource specified.
Specifying the sort_priority keyword sorts vnodes on the vnode priority attribute.
The default third argument for a resource is total. If the third argument, total | assigned | unused, is not specified with a resource, total is used. This provides backwards compatibility with previous releases.
The values used for sorting must be numerical.

4.8.48.2 Configuring Sorting Vnodes on a Key

You can specify up to 20 sort keys, where you want a primary sort key, a secondary sort key, etc.

If you specify more than one entry for node_sort_key, the first entry is the primary sort key, the second entry is the secondary sort key, which is used to sort equal-valued entries from the first sort, and so on.

Each entry is specified one to a line.

To sort jobs on a key, set the node_sort_key scheduler parameter:

• Set the desired key
• Specify whether high or low results should come first
• For sorting on a resource, optionally specify total, assigned, or unused
• Specify the primetime behavior

The scheduler’s configuration file is read on startup and HUP.
The argument to the `node_sort_key` parameter is a quoted string. The default for `node_sort_key` is the following:

```
node_sort_key: "sort_priority HIGH" all
```

See “node_sort_key” on page 288 of the PBS Professional Reference Guide.

### 4.8.48.2.i Examples of Sorting Vnodes

**Example 4-31:** This sorts vnodes by the highest number of unused CPUs:

```
node_sort_key: "ncpus HIGH unused" all
```

**Example 4-32:** This sorts vnodes by the highest amount of memory assigned to vnodes, but only during primetime:

```
node_sort_key: "mem HIGH assigned" prime
```

**Example 4-33:** This sorts vnodes according to speed. You want to run jobs on the fastest host available. You have 3 machines, where HostA is fast, HostB is medium speed, and HostC is slow.

Set node priorities so that faster machines have higher priority:

```
Qmgr: set node HostA priority = 200
Qmgr: set node HostB priority = 150
Qmgr: set node HostC priority = 100
```

Specify that vnodes are sorted according to priority, with highest priority first:

```
node_sort_key: "sort_priority HIGH" ALL
```

**Example 4-34:** The old “nodepack” behavior can be achieved by this:

```
node_sort_key: "ncpus low unused"
```

**Example 4-35:** In this example of the interactions between placement sets and `node_sort_key`, we have 8 vnodes numbered 1-8. The vnode priorities are the same as their numbers. However, in this example, when unsorted, the vnodes are selected in the
order 4, 1, 3, 2, 8, 7, 5, 6. This is to illustrate the change in behavior due to node_sort_key.

We use:

node_sort_key: "sort_priority LOW"

Using node_sort_key, the vnodes are sorted in order, 1 to 8. We have three placement sets:

A: 1, 2, 3, 4 when sorted by node_sort_key; 4, 1, 3, 2 when no node_sort_key is used
B: 5, 6, 7, 8 when sorted by node_sort_key; 8, 7, 5, 6 when no node_sort_key is used
C: 1-8 when sorted, 4, 1, 3, 2, 8, 7, 5, 6 when not sorted.

A 6-vnode job will not fit in either A or B, but will fit in C. Without the use of node_sort_key, it would get vnodes 4, 1, 3, 2, 8, 7. With node_sort_key, it would get vnodes 1 - 6, still in placement set C.

4.8.48.2.ii Caveats for Sorting Vnodes

- Sorting on a resource with node_sort_key and using "unused" or "assigned" cannot be used with load_balancing. If both are used, load balancing will be disabled.
- Sorting on a resource and using "unused" or "assigned" cannot be used with smp_cluster_dist when it is set to anything but "pack". If both are used, smp_cluster_dist will be set to "pack".
Chapter 5

PBS Resources

This chapter covers PBS resources, including providing resources for user jobs, setting up resources such as application licenses and scratch space, and how resources are used, defined, inherited, and viewed.

The PBS Professional Reference Guide contains resource reference material. For a list of built-in and custom Cray resources, as well as information on using resources, see “Resources” on page 297 of the PBS Professional Reference Guide. For a description of the format of each type of resource, see “Formats” on page 399 of the PBS Professional Reference Guide.

5.1 Introduction

PBS resources represent things such as CPUs, memory, application licenses, switches, scratch space, and time. They can also represent whether or not something is true, for example, whether a machine is dedicated to a particular project. PBS provides a set of built-in resources, and allows you to define additional custom resources. For some systems, PBS creates specific custom resources; see “Custom Cray Resources” on page 308 of the PBS Professional Reference Guide. The scheduler matches requested resources with available resources, according to rules defined by the administrator. PBS can enforce limits on resource usage by jobs. The administrator can specify which resources are available at the server, each queue, and each vnode.
# Chapter Contents

5.1 Introduction .................................................. 287
5.2 Chapter Contents ............................................ 288
5.3 Terminology ................................................... 289
5.4 Categories of Resources ........................................ 293
  5.4.1 Built-in and Custom Resources .......................... 293
  5.4.2 Server, Queue, and Vnode Resources ..................... 293
  5.4.3 Consumable and Non-consumable Resources ............. 294
  5.4.4 Static and Dynamic Resources ............................ 294
  5.4.5 Global and Local Resources ............................... 295
  5.4.6 Requested and Default Resources ......................... 296
  5.4.7 Shared and Non-shared Vnode Resources ................. 296
  5.4.8 Platform-specific and Generally Available Resources .. 296
  5.4.9 Job-wide and Chunk Resources ......................... 297
5.5 Resource Types ................................................ 298
5.6 Behavior of Resources ........................................ 298
  5.6.1 Default Behavior ........................................... 298
  5.6.2 How the Scheduler Uses Resources ...................... 300
  5.6.3 Resource Names ............................................. 300
  5.6.4 Resource Values ............................................ 300
5.7 How to Set Resource Values .................................... 300
  5.7.1 Editing Configuration Files Under Windows .............. 300
  5.7.2 Setting Values for Global Static Resources ............. 301
  5.7.3 Setting Values for Local Static Resources ............... 301
  5.7.4 Setting Values for String Arrays ......................... 301
  5.7.5 Resource Value Caveats .................................... 302
5.8 Overview of Ways Resources Are Used ....................... 303
  5.8.1 Advice on Using Resources ................................. 304
5.9 Resources Allocated to Jobs and Reservations ............... 304
  5.9.1 Allocating Chunks .......................................... 305
  5.9.2 Resources Requested by Job ............................... 305
  5.9.3 Specifying Job Default Resources ....................... 305
  5.9.4 Allocating Default Resources to Jobs .................... 309
  5.9.5 Dynamic Resource Allocation Caveats .................... 313
  5.9.6 Period When Resource is Used by Job .................... 313
5.10 Using Resources to Track and Control Allocation ........... 314
5.11 Using Resources for Topology and Job Placement ........... 317
  5.11.1 Restrictions on Using Resources for Job Placement ...... 317
5.12 Using Resources to Prioritize Jobs .......................... 317
5.13 Using Resources to Restrict Server, Queue Access .......... 318
5.3 Terminology

Advance reservation

A reservation for a specific set of resources for a specified start time and duration in the future. Advance reservations are created by users to reserve
resources for jobs. The reservation is available only to the creator of the reservation and any users or groups specified by the creator.

**Borrowing vnode**

A shared vnode resource is available for use by jobs at more than one vnode, but is managed at just one vnode. A *borrowing vnode* is a vnode where a shared vnode resource is available, but not managed.

**Built-in resource**

A resource that is defined in PBS Professional as shipped. Examples of built-in resources are `ncpus`, which tracks the number of CPUs, and `mem`, which tracks memory. See section 5.4.1, “Built-in and Custom Resources”, on page 293.

**Chunk**

A set of resources allocated as a unit to a job. Specified inside a selection directive. All parts of a chunk come from the same host. In a typical MPI (Message-Passing Interface) job, there is one chunk per MPI process.

**Consumable resource**

A consumable resource is a resource that is reduced or taken up by being used. Examples of consumable resources are memory or CPUs. See section 5.4.3, “Consumable and Non-consumable Resources”, on page 294.

**CPU**

Has two meanings, one from a hardware viewpoint, and one from a software viewpoint:

1. A core. The part of a processor that carries out computational tasks. Some systems present virtual cores, for example in hyperthreading.

2. Resource required to execute a program thread. PBS schedules jobs according, in part, to the number of threads, giving each thread a core on which to execute. The resource used by PBS to track CPUs is called “`ncpus`”. The number of CPUs available for use defaults to the number of cores reported by the OS. When a job requests one CPU, it is requesting one core on which to run.

**Custom resource**

A resource that is not defined in PBS as shipped. Custom resources are created by the PBS administrator or by PBS for some systems. See section 5.4.1, “Built-in and Custom Resources”, on page 293.

**Floating license**

A unit of license dynamically allocated (checked out) when a user begins using an application on some host (when the job starts), and deallocated (checked in) when a user finishes using the application (when the job ends).
Generic group limit
A limit that applies separately to groups at the server or a queue. This is the limit for groups which have no individual limit specified. A limit for generic groups is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Generic user limit
A limit that applies separately to users at the server or a queue. This is the limit for users who have no individual limit specified. A separate limit for generic users can be specified at the server and at each queue.

Global resource
A global resource is defined in a resources_available attribute, at the server, a queue, or a host. Global resources can be operated on via the qmgr command and are visible via the qstat and pbsnodes commands. See section 5.4.5, “Global and Local Resources”, on page 295.

Group limit
Refers to configurable limits on resources and jobs. This is a limit applied to the total used by a group, whether the limit is a generic group limit or an individual group limit.

Indirect resource
A shared vnode resource at vnode(s) where the resource is not defined, but which share the resource.

Individual group limit
Applies separately to groups at the server or a queue. This is the limit for a group which has its own individual limit specified. An individual group limit overrides the generic group limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Individual user limit
Applies separately to users at the server or a queue. This is the limit for users who have their own individual limit specified. A limit for an individual user overrides the generic user limit, but only in the same context, for example, at a particular queue. A separate limit can be specified at the server and each queue.
Chapter 5  

PBS Resources

**Limit**

A maximum that can be applied in various situations:

- The maximum amount of a resource that can be consumed at any time by running jobs.
- The maximum amount of a resource that can be allocated to queued and running jobs.
- The maximum number of jobs that can be running.
- The maximum number of jobs that can be queued.

**Local resource**

A local resource is defined in a Version 1 MoM configuration file. Local resources cannot be operated on via the `qmgr` command and are not visible via the `qstat` and `pbsnodes` commands. Local resources can be used by the scheduler. See section 5.4.5, “Global and Local Resources”, on page 295.

**Managing vnode**

The vnode where a shared vnode resource is defined, and which manages the resource.

**Memory-only vnode**

 Represents a node board that has only memory resources (no CPUs), for example, an Altix memory-only blade.

**Non-consumable resource**

A non-consumable resource is a resource that is not reduced or taken up by being used. Examples of non-consumable resources are Boolean resources and `walltime`. See section 5.4.3, “Consumable and Non-consumable Resources”, on page 294.

**Overall limit**

Limit on the total usage. In the context of server limits, this is the limit for usage at the PBS complex. In the context of queue limits, this is the limit for usage at the queue. An overall limit is applied to the total usage at the specified location. Separate overall limits can be specified at the server and each queue.

**Resource**

A resource can be something used by a job, such as CPUs, memory, high-speed switches, scratch space, licenses, or time, or it can be an arbitrary item defined for another purpose. PBS has built-in resources, and allows custom-defined resources.

**Shared resource**

A vnode resource defined and managed at one vnode, but available for use at others.
User limit  

Refers to configurable limits on resources and jobs. A user’s limit, whether generic or individual.

5.4 Categories of Resources

5.4.1 Built-in and Custom Resources

Built-in resources are the resources that are already defined for you in PBS. PBS supplies built-in resources including number of cpus, CPU time, and memory. For a list of built-in resources, see “Built-in Resources” on page 299 of the PBS Professional Reference Guide. Custom resources are those that you define, or that PBS creates for some systems. For example, if you wanted a resource to represent scratch space, you could define a resource called Scratch, and specify a script which queries for the amount of available scratch space. See section 5.14, “Custom Resources”, on page 319.

5.4.2 Server, Queue, and Vnode Resources

PBS resources can be available at the server, queues, both the server and queues, or at vnodes. Any of these resources can be static or dynamic, built-in or custom, and consumable or non-consumable. Vnode resources can additionally be global or local.

5.4.2.1 Server Resources

A server resource, also called a server-level resource, is a resource that is available at the server. A server resource is available to be consumed or matched at the server if you set the server’s resources_available.<resource name> attribute to the available or matching value. For example, you can define a custom resource called FloatingLicenses and set the server’s resources_available.FloatingLicenses attribute to the number of available floating licenses.

A server resource is a job-wide resource. This means that a job can request this resource for the entire job, but not for individual chunks.

An example of a job-wide resource is shared scratch space, or any custom resource that is defined at the server and queue level.

5.4.2.2 Queue Resources

A queue resource, also called a queue-level resource, is available to be consumed or matched by jobs in the queue if you set the queue’s resources_available.<resource name> attribute to the available or matching value.
A queue resource is a job-wide resource. A job can request a queue resource for the entire job, but not for individual chunks.

An example of a job-wide resource is floating licenses, or any custom resource that is defined at both server and queue level.

5.4.2.3 Resources Defined at Both Server and Queue

Custom resources can be defined to be available either at vnodes or at both the server and queues. Consumable custom resources that are defined at the server and queue level have their consumption monitored at the server and queue level. In our example, if a job requests one FloatingLicenses, then the value of the resources_assigned.FloatingLicenses attribute is incremented by one at both the server and the queue in which the job resides.

5.4.2.4 Vnode Resources

A vnode resource, also called a vnode-level or host-level resource, is available only at vnodes. A vnode resource is a chunk-level resource, meaning that it can be requested for a job only inside of a chunk.

5.4.3 Consumable and Non-consumable Resources

A consumable resource is one that is reduced by being used. Consumable resources include ncpus, mem and vmem by default, and any custom resource defined with the -n or -f flags.

A non-consumable resource is not reduced through use, meaning that allocation to one job does not affect allocation to other jobs. The scheduler matches jobs to non-consumable resources. Examples of non-consumable resources are walltime, file, cput, pcput, pmem, pvmem, nice, or Boolean resources.

5.4.4 Static and Dynamic Resources

Static resources are managed by PBS and have values that are fixed until you change them or until you change the hardware and MoM reports a new value for memory or number of CPUs.

Dynamic resources are not under the control of PBS, meaning that they can change independently of PBS. Dynamic resources are reported via a script; PBS runs a query to discover the available amount. Server dynamic resources use a script that runs at the server’s host. Host-level (MoM) dynamic resources use a script that runs at the execution host.

Static and dynamic resources can be available at the server or host level.
5.4.4.1 Dynamic Resource Caveats

Dynamic resource values are displayed in `qstat`, however, the value displayed is the last value retrieved, not the current value. Dynamic resources have no `resources_available.<resource>` representation anywhere in PBS.

5.4.5 Global and Local Resources

5.4.5.1 Global Static Resources

Global static resources are defined in `resources_available` attributes at the server, queue, or vnode, and are available at the server, queue, or vnode level. Global static resources can be operated on via the `qmgr` command and viewed via the `qstat` command. Values for built-in global static resources are set via the `qmgr` command. The `walltime` and `aoe` resources are examples of global static resources. For custom global static resources, see section 5.14.2.7, “Example of Defining Each Type of Custom Resource”, on page 330.

5.4.5.2 Global Dynamic Resources

Global dynamic resources are defined in the server’s `resource_def` file, and can be used at the server, queue, or vnode level. Global host-level dynamic resources can be viewed via the `qstat` command. Server dynamic resource values have no `resources_available.<resource>` representation anywhere in PBS. See section 5.14.4.1, “Dynamic Server-level Resources”, on page 334.

The value displayed via `qstat` for a dynamic resource is the most recently retrieved, not the current value.

5.4.5.3 Local Static Resources

It is not recommended to use local static resources. Local static resources are defined in the MoM Version 1 configuration file. These resources cannot be operated on via the `qmgr` command or viewed via the `qstat` command. They can be used by the scheduler.

5.4.5.4 Local Dynamic Resources

Dynamic local resources are defined in the MoM Version 1 configuration file. These are scripts that run on the execution host where they are defined and return a value. These resources can be used by the scheduler. Host dynamic resource values have no `resources_available.<resource>` representation anywhere in PBS. See section 5.14.5.1, “Dynamic Host-level Resources”, on page 337.
Chapter 5  PBS Resources

The value displayed via `qstat` for a dynamic resource is the most recently retrieved, not the current value.

### 5.4.6 Requested and Default Resources

A job’s requested resources are the resources explicitly requested by the job. Default resources are resources that you specify that each job should have if not requested. For example, you can specify that any job that does not request `walltime` gets 12 hours of `walltime`. For jobs that do request `walltime`, the default of 12 hours is not applied.

For information on default resources, see section 5.9.3, “Specifying Job Default Resources”, on page 305 and section 5.9.4, “Allocating Default Resources to Jobs”, on page 309.

### 5.4.7 Shared and Non-shared Vnode Resources

#### 5.4.7.1 Non-shared Vnode Resources

Most vnode resources are not shared. When a resource is defined at one vnode for use by jobs only at that vnode, the resource is not shared. For example, when `resources_available.ncpus` is set to 4 on a single-vnode machine, and no other vnodes have `resources_available.ncpus` defined as a pointer to this resource, this resource is not shared.

#### 5.4.7.2 Shared Vnode Resources

When more than one vnode needs access to the same actual resource, that resource can be shared among those vnodes. The resource is defined at one vnode, and the other vnodes that supply the resource contain a pointer to that vnode. Any of the vnodes can supply that resource to a job, but only up to the amount where the total being used by jobs is less than or equal to the total available at the vnode where the resource is defined. For example, if you had a 4-vnode machine which had 8GB of memory, and wanted any single vnode to be able to supply up to 8GB to jobs, you would make the memory a shared resource. See section 5.14.5.3, “Shared Host-level Resources”, on page 339.

### 5.4.8 Platform-specific and Generally Available Resources

Most PBS built-in resources are available on, and apply to, all supported platforms. However, PBS provides some resources specifically designed for a given platform. These platform-specific resources are not applicable to any other platform, and cannot be used on platforms other
than the one(s) for which they are designed. For example, PBS creates custom resources that represent Cray elements, such as the Cray `nid` and the Cray `label`. PBS has several built-in resources whose names begin with `mpp`; these apply only to the Cray.

5.4.9 Job-wide and Chunk Resources

5.4.9.1 Job-wide Resources

A job-wide resource applies to the entire job, and is available at the server or queue, but not at the host level. Job-wide resources are requested outside of a select statement, using this form:

`-l <resource name>=<value>`

For example, to request one hour of `walltime` for a job:

`-l walltime=1:00:00`

Examples of job-wide resources are `walltime`, scratch space, and licenses.

5.4.9.2 Chunk Resources

A chunk resource applies to the part of the job running on that chunk, and is available at the host level. Chunk resources are requested inside a select statement. A single chunk is requested using this form:

`-l select=<resource name>=<value>:<resource name>=<value>`

For example, one chunk might have 2 CPUs and 4GB of memory:

`-l select=ncpus=2:mem=4gb`

To request multiples of a chunk, prefix the chunk specification by the number of chunks:

`-l select=[number of chunks]<chunk specification>`

For example, to request six of the previous chunk:

`-l select=6:ncpus=2:mem=4gb`

To request different chunks, concatenate the chunks using the plus sign (“+”):

`-l select=[number of chunks]<chunk specification>+[number of chunks]<chunk specification>`

For example, to request two kinds of chunks, one with 2 CPUs per chunk, and one with 8 CPUs per chunk, both kinds with 4GB of memory:

`-l select=6:ncpus=2:mem=4gb+3:ncpus=8:mem=4gb`
5.5 Resource Types

PBS supplies the following types of resources:

- Boolean
- duration
- float
- long
- size
- string
- string_array

See “List of Formats” on page 399 of the PBS Professional Reference Guide for a description of each resource type.

5.6 Behavior of Resources

5.6.1 Default Behavior

PBS automatically collects information about some resources and sets their initial values accordingly. If you explicitly set the value for a resource, that value is carried forth across Server restarts.

5.6.1.1 Default Behavior of Vnode Resources

PBS sets the value for certain resources at each vnode. This means that the value for the vnode’s `resources_available.<resource name>` attribute is set by PBS. The following table lists the vnode resources that are set automatically by PBS.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Initial Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch</td>
<td>Value reported by OS</td>
<td>Settable. If you unset the value, it remains unset until MoM is restarted.</td>
</tr>
</tbody>
</table>
## Table 5-1: Resources Set by PBS

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Initial Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td><strong>Short form of hostname in Mom vnode attribute</strong></td>
<td>Settable. If you unset the value, it remains unset until MoM is restarted.</td>
</tr>
<tr>
<td>mem</td>
<td><strong>Amount reported by OS</strong></td>
<td>Settable. If you unset the value, it remains unset until MoM is restarted.</td>
</tr>
<tr>
<td>ncpus</td>
<td><strong>Number of CPUs reported by OS</strong></td>
<td>Settable. If you unset this value, the MoM will reset it to the value reported by the OS.</td>
</tr>
<tr>
<td>PBScrayhost</td>
<td>On CLE 2.2, set to default. On CLE 3.0 and higher, set to value of mpp_host for this system</td>
<td>Do not set.</td>
</tr>
<tr>
<td>PBScraylabel_&lt;label name&gt;</td>
<td><strong>Concatenation of PBScraylabel_ and label name. Set to True on all of node s vnodes.</strong></td>
<td>Do not set.</td>
</tr>
<tr>
<td>PBScraynid</td>
<td><strong>Value of node_id for this compute node</strong></td>
<td>Do not set.</td>
</tr>
<tr>
<td>PBScrayorder</td>
<td>Value starts at 1 and increments by 1 for each node in inventory</td>
<td>Do not set.</td>
</tr>
<tr>
<td>PBScrayseg</td>
<td><strong>Segment ordinal of associated NUMA node.</strong></td>
<td>Do not set.</td>
</tr>
<tr>
<td>router</td>
<td><strong>Name of router, from topology file</strong></td>
<td>Applies to vnodes on certain Altix machines only, such as the 4700.</td>
</tr>
<tr>
<td>vnode</td>
<td><strong>Name of the vnode</strong></td>
<td>Vnode name must be specified via the qmgr create node command.</td>
</tr>
</tbody>
</table>
Chapter 5  

PBS Resources

For example, PBS automatically sets the value of resources_available.ncpus at each vnode.

5.6.1.2 Default Behavior of Server and Queue Resources

PBS automatically sets the value for default_chunk.ncpus to 1 at the server and queues.

5.6.1.3 Default Behavior of Job Resources

PBS automatically sets the value of the estimated.start_time job resource to the estimated start time for each job.

5.6.2 How the Scheduler Uses Resources

How the scheduler uses resources is described in section 4.8.28, “Matching Jobs to Resources”, on page 197.

5.6.3 Resource Names

Resource names are case-insensitive. See “Resource Name” on page 403 of the PBS Professional Reference Guide for the format of resource names.

5.6.4 Resource Values

String resource values are case-sensitive. For format information, see “Resource Value” on page 403 of the PBS Professional Reference Guide.

5.7 How to Set Resource Values

Since the value for each dynamic resource is set by PBS to the value returned by a script or command, you will set values for static resources only.

You set values for custom and built-in resources using the same methods.

5.7.1 Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.
5.7.2 Setting Values for Global Static Resources

To set the value for a global vnode, queue, or server resource, use the `qmgr` command to set the value of the appropriate `resources_available.<resource>` attribute.

Example 5-1: Set the value of `floatlicenses` at the server to 10:

```
qmgr: set server resources_available.floatlicenses = 10
```

Example 5-2: Set the value of `RunsMyApp` to `True` at the vnode named `Vnode1`:

```
qmgr: set node Vnode1 resources_available.RunsMyApp = True
```

5.7.2.1 Restrictions on Setting Values for Global Static Resources

When setting global static vnode resources on multi-vnode machines, follow the rules in section 3.5.2, “Choosing Configuration Method”, on page 52.

5.7.3 Setting Values for Local Static Resources

It is not recommended to use local static resources, because these resources cannot be requested, and cannot be viewed using `qstat` or managed using `qmgr`. To set the value of a local vnode resource, edit `PBS_HOME/mom_priv/config` and change the value section of the resource’s line.

5.7.4 Setting Values for String Arrays

A string array that is defined on vnodes can be set to a different set of strings on each vnode.
Example of defining and setting a string array:

- Define a new resource:
  
  ```
  foo_arr type=string_array flag=h
  ```

- Setting via `qmgr`:
  
  ```
  Qmgr: set node n4 resources_available.foo_arr="f1, f3, f5"
  ```

- Vnode n4 has 3 values of `foo_arr`: f1, f3, and f5. We add f7:
  
  ```
  Qmgr: set node n4 resources_available.foo_arr+=f7
  ```

- Vnode n4 now has 4 values of `foo_arr`: f1, f3, f5 and f7.

- We remove f1:
  
  ```
  Qmgr: set node n4 resources_available.foo_arr-=f1
  ```

- Vnode n4 now has 3 values of `foo_arr`: f3, f5, and f7.

- Submission:
  
  ```
  qsub -l select=1:ncpus=1:foo_arr=f3
  ```

### 5.7.5 Resource Value Caveats

- It is not recommended to set the value for `resources_available.ncpus`. The exception is when you want to oversubscribe CPUs. See section 9.4.5.1.iii, “How To Share CPUs”, on page 740.

- Do not attempt to set values for `resources_available.<resource>` for dynamic resources.

- Do not set values for any resources, except those such as shared scratch space or floating licenses, at the server or a queue, because the scheduler will not allocate more than the specified value. For example, if you set `resources_available.walltime` at the server to `10:00:00`, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.

#### 5.7.5.1 Resource Value Caveats for Multi-vnode Machines

- When setting global static vnode resources on multi-vnode machines, follow the rules in section 3.5.2, “Choosing Configuration Method”, on page 52.

- It is not recommended to change the value of `ncpus` at vnodes on a multi-vnoded machine.

- On multi-vnode machines, do not set the values for `mem`, `vmem` or `ncpus` on the natural vnode. If any of these resources has been explicitly set to a non-zero value on the natural vnode, set `resources_available.ncpus`, `resources_available.mem` and
resources_available.vmem to zero on each natural vnode.

• On the natural vnode, all values for `resources_available.<resource>` should be zero (0), unless the resource is being shared among other vnodes via indirection.

### 5.8 Overview of Ways Resources Are Used

Resources are used in several ways in PBS. The following table lists the ways resources are used, and gives links to the section describing each one:

<table>
<thead>
<tr>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation to and use by jobs</td>
<td>See section 5.9, “Resources Allocated to Jobs and Reservations”, on page 304</td>
</tr>
<tr>
<td>Limiting job resource usage</td>
<td>See section 5.15.3, “Placing Resource Limits on Jobs”, on page 391</td>
</tr>
<tr>
<td>Restricting access to server and queues</td>
<td>See section 5.13, “Using Resources to Restrict Server, Queue Access”, on page 318</td>
</tr>
<tr>
<td>Routing jobs</td>
<td>See section 2.2.6.4, “Using Resources to Route Jobs Between Queues”, on page 25</td>
</tr>
<tr>
<td>Describing topology and placing jobs</td>
<td>See section 5.11, “Using Resources for Topology and Job Placement”, on page 317</td>
</tr>
<tr>
<td>Setting job execution priority</td>
<td>See section 5.12, “Using Resources to Prioritize Jobs”, on page 317</td>
</tr>
<tr>
<td>Reserving resources ahead of time</td>
<td>See section 4.8.37, “Advance and Standing Reservations”, on page 249</td>
</tr>
<tr>
<td>Tracking and controlling allocation</td>
<td>See section 5.10, “Using Resources to Track and Control Allocation”, on page 314</td>
</tr>
<tr>
<td>Determining job preemption priority</td>
<td>See section 4.8.33, “Using Preemption”, on page 228</td>
</tr>
</tbody>
</table>
5.8.1 Advice on Using Resources

See “Advice on Using Resources” on page 297 of the PBS Professional Reference Guide for tips on using resources.

5.9 Resources Allocated to Jobs and Reservations

Resources allocated to jobs provide the job with items such as CPUs and memory to be consumed by the job’s processes, as well as qualities such as architecture and host. The resources allocated to a job are those that the job requests and those that are assigned to it through resource defaults that you define.

Jobs use resources at the job-wide and chunk level. Job-wide resources such as walltime or vmem are applied to and requested by the job as a whole. Chunk-level resources, such as ncpus, are applied and requested in individual chunks.

Jobs explicitly request resources either at the vnode level in chunks defined in a selection statement, or in job-wide resource requests. See “Resources” on page 297 of the PBS Professional Reference Guide and ”Requesting Resources”, on page 71 of the PBS Professional User’s Guide.

Jobs inherit resource defaults for resources not explicitly requested. See section 5.9.4, “Allocating Default Resources to Jobs”, on page 309.

Chunk-level resources are made available at the host (vnode) level by defining them via resources_available.<resource> at the vnode, and are requested using -l select=<resource>=<value>.

Job-wide resources are made available by defining them via resources_available.<resource> at the queue or server. These resources are requested using -l <resource> =<value>.

The scheduler matches requested resources with available resources, according to rules defined by the administrator.

When a job is requesting a string array resource, it can request only one of the values set in the string array resource. The job will only be placed on a vnode where the job’s requested string matches one of the values of the string array resource. For example, if the resource named Colors is set to “red, blue, green” on vnode V1, and “red, blue” on V2:

- A job can request only one of “red”, “blue”, or “green”
- A job requesting Colors=green will only be placed on V1
5.9.1 Allocating Chunks

Chunks cannot be split across hosts. Chunks can be made up of vchunks. If a chunk is broken up over multiple vnodes, all participating vnodes must belong to the same execution host. Each vnode supplies a vchunk. These participating vnodes are supplying the vchunks that make up the chunk. A chunk defines a logical set of resources, for example, those needed for an MPI task. The resources must come from a single host, but if the requested resources exceed that of any one vnode, the physical resources can be taken from multiple vnodes on the same host.

5.9.2 Resources Requested by Job

The job’s `Resource_List` attribute lists the following resources requested by the job:

- Job-wide resources either explicitly requested by the job or inherited from defaults
- The following built-in chunk-level resources either explicitly requested by the job or inherited from defaults:
  - `mpiprocs`
  - `ncpus`
  - `netwins`
  - `mem`
  - `vmem`
- Custom vnode-level (chunk-level) resources that are global and have the n, q, or f flags set, either explicitly requested by the job or inherited from defaults

5.9.3 Specifying Job Default Resources

You can specify which resources are automatically added to job resource requests. When a job does not request a specific resource, the default value for that resource is automatically added to the job’s resource request.

The amount of each resource a job is allowed to use is the amount in its resource request. See section 5.15.3, “Placing Resource Limits on Jobs”, on page 391. Therefore you may wish to add default limits on resource usage. This is done by adding default resources to the job’s resource request. For example, if a job does not request `walltime`, but you do not want jobs not specifying `walltime` to run for more than 12 hours, you can specify a default of 12 hours for `walltime`. Jobs that do specify `walltime` do not inherit this default; they keep their requested amount.
You can use default resources to manage jobs. For example, if you want to keep track of and limit the number of jobs using something such as a disk arm, you can have each job using the disk arm automatically request one counting resource. Then you can place a limit on the amount of this resource that can be in use at one time. This technique is described in section 5.10, “Using Resources to Track and Control Allocation”, on page 314.

Default resources can be defined for the server and for each queue. Default resources defined at the server are applied to all jobs. Default resources at a queue are applied only to the jobs that are in that queue.

Default resources on the server and queue can be job-wide, which is the same as adding -l <resource name> to the job’s resource request, or they can be chunk resources, which is the same as adding :<resource name>=<value> to a chunk.

Job-wide resources are specified via resources_default on the server or queue, and chunk resources are specified via default_chunk on the server or queue. You can also specify default resources to be added to any qsub arguments. In addition, you can specify default placement of jobs.

5.9.3.1 Specifying Job-wide Default Resources at Server

To specify a server-level job-wide default resource, use the qmgr command to set the server’s resources_default attribute:

```
Qmgr: set server resources_default.<resource>=<value>
```

For example, to set the default architecture on the server:

```
Qmgr: set server resources_default.arch=linux
```

5.9.3.2 Specifying Chunk Default Resources at Server

To specify a server-level chunk default resource, use the qmgr command to set the server’s default_chunk attribute:

```
Qmgr: set server default_chunk.<resource>=<value>
```

For example, if you want all chunks that don’t specify ncpus or mem to inherit the values you specify:

```
Qmgr: set server default_chunk.ncpus=1
Qmgr: set server default_chunk.mem=1gb
```
5.9.3.3 Specifying Job-wide Default Resources at Queue

To specify a default for a job-wide resource at a queue, use the qmgr command to set the queue’s resources_default attribute:

Qmgr: set queue <queue name> resources_default.<resource> = <value>

5.9.3.4 Specifying Chunk Default Resources at Queue

To specify a queue-level chunk default resource, use the qmgr command to set the queue’s default_chunk attribute:

Qmgr: set queue <queue name> default_chunk.<resource>=<value>

For example, if you want all chunks that don’t specify ncpus or mem to inherit the values you specify:

Qmgr: set queue small default_chunk.ncpus=1
Qmgr: set queue small default_chunk.mem=512mb

5.9.3.5 Specifying Default qsub Arguments

You can set defaults for any qsub arguments not explicitly requested by each job. You do this at the server by using the qmgr command to set the server’s default_qsub_arguments attribute:

Qmgr: set server default_qsub_arguments=<string containing arguments>

For example, to set the default for the Rerunable job attribute in each job’s resource request, and the name of the job:

Qmgr: set server default_qsub_arguments= "-r y -N MyJob"

Or to set a default Boolean in each job’s resource request so that jobs don’t run on Red unless they explicitly ask to do so:

Qmgr: set server default_qsub_arguments="-l Red=False"

5.9.3.6 Specifying Default Job Placement

You can specify job placement defaults at both the server and queue level. You use the qmgr command to set the resources_default.place attribute at the server or queue:

Qmgr: set queue <queue name> resources_default.place=<value>

For example, to set the default job placement for a queue:

Qmgr: set queue Q1 resources_default.place=free
When setting default placement involving a colon, enclose the value in double quotes:

```
Qmgr: set server resources_default.place="<value>"
```

For example, to set default placement at the server to `pack:shared`, do the following:

```
Qmgr: set server resources_default.place= "pack:shared"
```

See "Specifying Job Placement", on page 89 of the PBS Professional User’s Guide for detailed information about how `-l place` is used.

### 5.9.3.7 Using Gating Values As Defaults

For most resources, if the job does not request the resource, and no server or queue defaults are set, the job inherits the maximum gating value for the resource. If this is set at the queue, the queue value of `resources_max.<resource>` is used. If this is set only at the server, the job inherits the value set at the server. However, for `mpp` resources, the job does not inherit the gating value. For example, if the job does not request `mppnppn`, and no defaults are set at the server and queue, but `resources_max.mppnppn` is set at the queue, the job does not inherit the queue’s value.

### 5.9.3.8 Default Resource Caveats

- While users cannot request custom resources that are created with the `r` flag, jobs can inherit these as defaults from the server or queue `resources_default.<resource>` attribute.

- A `qsub` or `pbs_rsub` hook does not have resources inherited from the server or queue `resources_default` or `default_chunk` as an input argument.

- For `mpp` resources, the job does not inherit the gating value. For example, if the job does not request `mppnppn`, and no defaults are set at the server and queue, but `resources_max.mppnppn` is set at the queue, the job does not inherit the queue’s value.

- Default `qsub` arguments and server and queue defaults are applied to jobs at a coarse level. Each job is examined to see whether it requests a `select` and a `place`. This means that if you specify a default placement, such as `excl`, with `-lplace=excl`, and the user specifies an arrangement, such as `pack`, with `-lplace=pack`, the result is that the job ends up with `-lplace=pack`, NOT `-lplace=pack:excl`. The same is true for `select`; if you specify a default of `-lselect=2:ncpus=1`, and the user specifies `-lselect=mem=2GB`, the job ends up with `-lselect=mem=2GB`.
5.9.4 Allocating Default Resources to Jobs

Jobs inherit default resources, job-wide or per-chunk, with the following order of precedence.

**Table 5-3: Order In Which Default Resources Are Assigned to Jobs**

<table>
<thead>
<tr>
<th>Order of assignment</th>
<th>Default value</th>
<th>Affects Chunks?</th>
<th>Job-wide?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Default <code>qsub</code> arguments</td>
<td>If specified</td>
<td>If specified</td>
</tr>
<tr>
<td>2</td>
<td>Queue’s <code>default_chunk</code></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Server’s <code>default_chunk</code></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Queue’s <code>resources_default</code></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Server’s <code>resources_default</code></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>Queue’s <code>resources_max</code></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>Server’s <code>resources_max</code></td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See section 5.9.3, “Specifying Job Default Resources”, on page 305 for how to set these defaults.

For each chunk in the job’s selection statement, first default `qsub` arguments are applied, then queue chunk defaults are applied, then server chunk defaults are applied. If the chunk does not contain a resource defined in the defaults, the default is added. The chunk defaults are specified in the `default_chunk.<resource name>` server or queue attribute.

For example, if the queue in which the job is enqueued has the following defaults defined,

```
default_chunk.ncpus=1
default_chunk.mem=2gb
```

then a job submitted with this selection statement:

```
select=2:ncpus=4+1:mem=9gb
```

will have this specification after the `default_chunk` elements are applied:

```
select=2:ncpus=4:mem=2gb+1:ncpus=1:mem=9gb
```

In the above, `mem=2gb` and `ncpus=1` are inherited from `default_chunk`.
Chapter 5  PBS Resources

The job-wide resource request is checked against queue resource defaults, then against server resource defaults, then against the queue’s resources_max.<resource>, then against the server’s resources_max.<resource>. If a default or maximum resource is defined which is not specified in the resource request, it is added to the resource request.

5.9.4.1 Default Resource Allocation for min_walltime and max_walltime

The min_walltime and max_walltime resources inherit values differently. A job can inherit a value for max_walltime from resources_max.walltime; the same is not true for min_walltime. This is because once a job is shrink-to-fit, PBS can use a walltime limit for max_walltime. See section 4.8.41.3.ii, “Inheriting Values for min_walltime and max_walltime”, on page 265.

5.9.4.2 Default Resource Allocation Caveats

• Resources assigned from the default_qsub_arguments server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag, whether that resource was requested by the user or came from default_qsub_arguments. Be aware that creating custom resources with permission flags and then using these in the default_qsub_arguments server attribute can cause jobs to be rejected. See section 5.14.2.6, “Resource Permission Flags”, on page 328.

• Default qsub arguments and server and queue defaults are applied to jobs at a coarse level. Each job is examined to see whether it requests a select and a place. This means that if you specify a default placement, such as excl, with -lplace=excl, and the user specifies an arrangement, such as pack, with -lplace=pack, the result is that the job ends up with -lplace=pack, NOT -lplace=pack:excl. The same is true for select; if you specify a default of -lselect=2:ncpus=1, and the user specifies -lselect=mem=2GB, the job ends up with -lselect=mem=2GB.

5.9.4.3 Moving Jobs Between Queues or Servers Changes Defaults

If the job is moved from the current queue to a new queue or server, any default resources in the job’s Resource_List inherited from the current queue or server are removed. The job then inherits any new default resources. This includes a select specification and place directive generated by the rules for conversion from the old syntax. If a job's resource is unset (undeclared) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either select or place is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.
Jobs may be moved between servers when peer scheduling is in operation. Given the following set of queue and server default values:

- **Server**
  
  resources_default.ncpus=1

- **Queue QA**
  
  resources_default.ncpus=2  
  default_chunk.mem=2GB

- **Queue QB**
  
  default_chunk.mem=1GB  
  no default for ncpus
The following illustrate the equivalent select specification for jobs submitted into queue QA and then moved to (or submitted directly to) queue QB:

Example 5-3: Submission:
```
qsub -l ncpus=1 -lmem=4gb
```
- In QA:
  ```
  select=1:ncpus=1:mem=4gb
  ```
  - No defaults need be applied
- In QB:
  ```
  select=1:ncpus=1:mem=4gb
  ```
  - No defaults need be applied

Example 5-4: Submission:
```
qsub -l ncpus=1
```
- In QA:
  ```
  select=1:ncpus=1:mem=2gb
  ```
  - Picks up 2GB from queue default chunk and 1 ncpus from qsub
- In QB:
  ```
  select=1:ncpus=1:mem=1gb
  ```
  - Picks up 1GB from queue default_chunk and 1 ncpus from qsub

Example 5-5: Submission:
```
qsub -l mem=4gb
```
- In QA:
  ```
  select=1:ncpus=2:mem=4gb
  ```
  - Picks up 2 ncpus from queue level job-wide resource default and 4GB mem from qsub
- In QB:
  ```
  select=1:ncpus=1:mem=4gb
  ```
  - Picks up 1 ncpus from server level job-wide default and 4GB mem from qsub

Example 5-6: Submission:
```
qsub -l nodedes=4
```
- In QA:
  ```
  select=4:ncpus=1:mem=2gb
  ```
  - Picks up a queue level default memory chunk of 2GB. (This is not 4:ncpus=2 because
in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)

- In QB:
  \texttt{select=4:ncpus=1:mem=1gb}

  (In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the \texttt{ncpus=1} is not inherited from the server default.)

Example 5-7: Submission:

\texttt{qsub -l mem=16gb -lnodes=4}

- In QA:
  \texttt{select=4:ncpus=1:mem=4gb}

  (This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)

- In QB:
  \texttt{select=4:ncpus=1:mem=4gb}

  (In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the \texttt{ncpus=1} is not inherited from the server default.)

5.9.5 Dynamic Resource Allocation Caveats

When a job requests a dynamic resource, PBS checks to see how much of the resource is available, but cannot know how much will be used by another job while this job executes. This can lead to a resource shortage. For example, there is 20GB of scratch on a disk, no jobs are running, and a job requests 15GB. This job writes to 5GB during the first part of its execution, then another job requests 10GB. The second job is started by PBS, because there is 15GB available. Now there is a shortage of scratch space.

You can avoid this problem by configuring a static consumable resource to represent scratch space. Set it to the amount of available scratch space. See section 5.14.6.3, “Static Server-level Scratch Space”, on page 346 and section 5.14.6.4, “Static Host-level Scratch Space”, on page 346.

5.9.6 Period When Resource is Used by Job

5.9.6.1 Exiting Job Keeps Resource

A job that is exiting is still consuming resources assigned to it. Those resources are available for other jobs only when the job is finished.
5.9.6.2  Job Suspension and Resource Usage

When a job is suspended, PBS releases all of the job's resources, including the licenses used by PBS for the job. This does not include the licenses used by the application, if any.

Jobs are suspended, and release their licenses, for preemption, and via `qsig -s suspend`.

A job is resumed only when sufficient resources are available. When a person resumes a job, the job is not run until resources are available.

5.9.6.2.i  Suspension/resumption Resource Caveats

Dynamic resources can cause problems with suspension and resumption of jobs.

When a job is suspended, its resources are freed, but the scratch space written to by the job is not available.

A job that uses scratch space may not suspend and resume correctly. This is because if the job writes to scratch, and is then suspended, when PBS queries for available scratch to resume the job, the script may return a value too small for the job's request. PBS cannot determine whether the job itself is the user of the scratch space; PBS can only determine how much is still unused. If a single suspended job has left less scratch space available than it requests, that job cannot be resumed.

The above is true for any dynamic resource, such as application licenses.

5.9.6.3  Shrink-to-fit Jobs Get walltime When Executed

PBS computes the `walltime` value for each shrink-to-fit job when the scheduler runs the job, not before. See section 4.8.41.3.iii, “Setting walltime for Shrink-to-fit Jobs”, on page 265.

5.10  Using Resources to Track and Control Allocation

You can use resources to track and control usage of things like hardware and licenses. For example, you might want to limit the number of jobs using floating licenses or a particular vnode. There is more than one way to accomplish this.

Example 5-8: You can set a complex-wide limit on the number of jobs using a type of complex-wide floating license. This example uses a single queue for the entire complex.
This method requires job submitters to request one of a `floatlicensecount` resource in order to be able to use the license. To set a complex-wide limit, take the following steps:

1. Create a custom static integer license resource that will be tracked at the server and queue:
   a. In `PBS_HOME/server_priv/resourcedef`, add the line:
      
      ```
      floatlicensecount   type=long   flag=q
      ```
   b. Add the resource to the `resources` line in `PBS_HOME/sched_priv/sched_config`:
      
      ```
      resources: "[...], floatlicensecount"
      ```


3. HUP the scheduler:

   ```
   kill -HUP <scheduler PID>
   ```

4. Set the available resource at the server using `qmgr`. If you have enough floating licenses for 4 jobs:

   ```
   qmgr: set server resources_available.floatlicensecount = 4
   ```

5. Inform job submitters that jobs using they must request one job-wide `floatlicensecount` resource via the following:

   ```
   qsub -l floatlicensecount=1
   ```

   The scheduler will schedule up to 4 jobs at a time using the licenses. You do not need to set the resource at any queue.

Example 5-9: Here, your job submitters don’t need to request a counting resource. Jobs are routed based on the size of the request for memory, and the counting resource is inherited
from a default. In this example, we are limiting the number of jobs from each group that can use a particular vnode that has a lot of memory. This vnode is called MemNode.

Jobs that request 8GB or more of memory are routed into queue BigMem, and inherit a default counting resource called memcount. All other jobs are routed into queue SmallMem. The routing queue is called RouteQueue.

1. Create a custom static integer memcount resource that will be tracked at the server and queue:
   a. In PBS_HOME/server_priv/resourcedef, add the line:
      memcount type=long flag=q
   b. Add the resource to the resources: line in PBS_HOME/sched_priv/
      sched_config:
         resources: "[...], memcount"
3. HUP the scheduler:
   kill -HUP <scheduler PID>
4. Set limits at BigMem and SmallMem so that they accept the correct jobs:
   Qmgr: set queue BigMem resources_min.mem = 8gb
   Qmgr: set queue SmallMem resources_max.mem = 8gb
5. Set the order of the destinations in the routing queue so that BigMem is tested first, so that jobs requesting exactly 8GB go into BigMem:
   Qmgr: set queue RouteQueue route_destinations = “BigMem, SmallMem”
6. Set the available resource at BigMem using qmgr. If you want a maximum of 6 jobs from BigMem to use MemNode:
   Qmgr: set queue BigMem resources_available.memcount = 6
7. Set the default value for the counting resource at BigMem, so that jobs inherit the value:
   Qmgr: set queue BigMem resources_default.memcount = 1
8. Associate the vnode with large memory with the BigMem queue:
   Qmgr: set node MemNode queue = BigMem
   The scheduler will only schedule up to 6 jobs from BigMem at a time on the vnode with large memory.
5.11 Using Resources for Topology and Job Placement

Using the topology information in the server’s node_group_key attribute, PBS examines the values of resources at vnodes, and uses those values to create placement sets. Jobs are assigned to placement sets according to their resource requests. Users can specify particular placement sets by requesting the resources that define that particular placement set. For example, if the switch named A25 connects the desired set of vnodes, a user can request the following:

-1 switch=A25

See section 4.8.32, “Placement Sets”, on page 211.

5.11.1 Restrictions on Using Resources for Job Placement

Only vnode-level resources can be used to direct jobs to particular vnodes.

5.12 Using Resources to Prioritize Jobs

You can define the formula the scheduler uses to compute job execution priorities. Elements in this formula can be inherited default custom resources. These resources must be job-wide numeric resources, or consumable host-level resources. See section 5.9.3, “Specifying Job Default Resources”, on page 305 and section 4.8.20, “Using a Formula for Computing Job Execution Priority”, on page 182.

You can make jobs inherit numeric resources according to non-numeric qualities, such as the job owner’s group or whether the job requests a Boolean or string resource. You can do this by either of the following methods:

- Use a hook to identify the jobs you want and alter their resource requests to include the custom resources for the formula. See Chapter 6, “Hooks”, on page 413
- Use a routing queue and minimum and maximum resource limits to route jobs to queues where they inherit the default custom resources for the formula. See section 2.2.6.4, “Using Resources to Route Jobs Between Queues”, on page 25

For details on how job execution priority is calculated, see section 4.8.16, “Calculating Job Execution Priority”, on page 166.

For a complete description of how PBS prioritizes jobs, see section 4.2.5, “Job Prioritization”, on page 67.
5.13 Using Resources to Restrict Server, Queue Access

You can set resource limits at the server and queues so that jobs must conform to the limits in order to be admitted. This way, you can reject jobs that request more of a resource than the complex or a queue can supply. You can also force jobs into specific queues where they will inherit the desired values for unrequested or custom resources. You can then use these resources to manage jobs, for example by using them in the job sorting formula or to route jobs to particular vnodes.

You set a maximum for each resource at the server using the `resources_max.<resource>` server attribute; there is no `resources_min.<resource>` at the server.

You can set a minimum and a maximum for each resource at each queue using the `resources_min.<resource>` and `resources_max.<resource>` queue attributes.

Job resource requests are compared to resource limits the same way, whether at the server or a queue. For a complete description of how jobs are tested against limits, see section 2.2.6.4.i, “How Queue and Server Limits Are Applied, Except Running Time”, on page 26.

Job resource requests are compared first to queue admittance limits. If there is no queue admittance limit for a particular resource, the job’s resource request is compared to the server’s admittance limit.

5.13.1 Admittance Limits for walltime, min_walltime, and max_walltime

Because `min_walltime` and `max_walltime` are themselves limits, they behave differently from other time-based resources. When a shrink-to-fit job (a job with a value for `min_walltime`) is compared to server or queue limits, the following must be true in order for the job to be accepted:

- Both `min_walltime` and `max_walltime` must be greater than or equal to `resources_min.walltime`.
- Both `min_walltime` and `max_walltime` must be less than or equal to `resources_max.walltime`.

You cannot set `resources_min` or `resources_max` for `min_walltime` or `max_walltime`. 
5.13.2 Restrictions on Resources Used for Admittance

For a list of resources that are compared to admittance limits, see section 2.2.6.4.iii, “Resources Used for Routing and Admittance”, on page 27. For information on using strings, string arrays, and Booleans for admittance controls, see section 2.2.6.4.iv, “Using String, String Array, and Boolean Values for Routing and Admittance”, on page 27.

5.14 Custom Resources

You can define, that is, create, new resources within PBS. The primary use of this feature is to add site-specific resources, such as to manage software application licenses. This section describes how to define and use custom resources.

Once new resources are defined, jobs may request these new resources and the scheduler can schedule on the new resources.

Using this feature, it is possible to schedule resources where the number or amount available is outside of PBS's control.

Custom resources can be made invisible to users or unalterable by users via resource permission flags. See section 5.14.2.6, “Resource Permission Flags”, on page 328. A user will not be able to print or list custom resources which have been made either invisible or unalterable.

PBS provides certain custom resources that are designed to reflect resources or properties found on specific systems. Do not create custom resources with the names that PBS uses for these resources. See “Custom Cray Resources” on page 308 of the PBS Professional Reference Guide.

5.14.1 How to Use Custom Resources

Custom resources can be static or dynamic, server-level or host-level, and local or global. They can also be shared or not.

5.14.1.1 Choosing the Resource Category

Use dynamic resources for quantities that PBS does not control, such as externally-managed licenses or scratch space. PBS runs a script or program that queries an external source for the amount of the resource available and returns the value via stdout. Use static resources for things PBS does control, such as licenses managed by PBS. PBS tracks these resources internally.
Use server-level resources for things that are not tied to specific hosts, that is, they can be available to any of a set of hosts. An example of this is a floating license. Use host-level resources for things that are tied to specific hosts, like the scratch space on a machine or node-locked licenses.

### 5.14.1.1.i Quick Guide to Configuring a Custom Resource

The following table gives a quick guide to configuring each kind of custom resource:

<table>
<thead>
<tr>
<th>Use for Resource</th>
<th>Link to Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>License: Floating, externally-managed</td>
<td>See section 5.14.7.3.i, “Example of Floating, Externally-managed License”, on page 348</td>
</tr>
<tr>
<td>License: Floating, externally-managed with features</td>
<td>See section 5.14.7.3.ii, “Example of Floating, Externally-managed License with Features”, on page 350</td>
</tr>
<tr>
<td>License: Floating, PBS-managed</td>
<td>See section 5.14.7.3.iii, “Example of Floating License Managed by PBS”, on page 351</td>
</tr>
<tr>
<td>License: Node-locked, per-host</td>
<td>See section 5.14.7.4.iv, “Example of Per-host Node-locked Licensing”, on page 354</td>
</tr>
<tr>
<td>License: Node-locked, per-CPU</td>
<td>See section 5.14.7.4.v, “Example of Per-CPU Node-locked Licensing”, on page 358</td>
</tr>
<tr>
<td>License: Node-locked, per-use</td>
<td>See section 5.14.7.4.v, “Example of Per-use Node-locked Licensing”, on page 356</td>
</tr>
<tr>
<td>GPU: specific GPU</td>
<td>See “Configuring PBS for Advanced GPU Scheduling” on page 362</td>
</tr>
</tbody>
</table>
5.14.1.2 Dynamic Custom Resources

A dynamic resource is one which is not under the control of PBS, meaning it can change independently of PBS. In order to use a dynamic resource, PBS must run a query to discover the available amount of that resource. Dynamic custom resources can be defined at the server or vnodes.

5.14.1.2.i Dynamic Server-level Custom Resources

A dynamic server-level custom resource is used to track a resource that is available on the server/scheduler host. You use a dynamic server-level resource to track something that is not under the control of PBS, and changes outside of PBS, for example, floating licenses. At each scheduler cycle, the scheduler runs a script at the scheduler/server host to determine the available amount of that resource. Server-level custom resources are used as job-wide resources.

5.14.1.2.ii Dynamic Host-level Custom Resources

A dynamic host-level custom resource is used to track a resource that is available at the execution host or hosts. You use a dynamic host-level resource for a resource that is not under the control of PBS, and changes outside of PBS, for example, scratch space. At each scheduler cycle, the scheduler queries the MoM for the available amount of the resource. The MoM runs a script which returns the current value of the resource. Host-level dynamic resources are used inside chunks.

<table>
<thead>
<tr>
<th>Use for Resource</th>
<th>Link to Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic dynamic server-level</td>
<td>See section 5.14.4.1.i, “Example of Configuring Dynamic Server-level Resource”, on page 335</td>
</tr>
<tr>
<td>Generic static server-level</td>
<td>See section 5.14.4.2.i, “Example of Configuring Static Server-level Resource”, on page 336</td>
</tr>
<tr>
<td>Generic dynamic host-level</td>
<td>See section 5.14.5.1.i, “Example of Configuring Dynamic Host-level Resource”, on page 338</td>
</tr>
<tr>
<td>Generic static host-level</td>
<td>See section 5.14.5.2.i, “Example of Configuring Static Host-level Resource”, on page 339</td>
</tr>
<tr>
<td>Generic shared static host-level</td>
<td>See section 5.14.5.3.v, “Configuring Shared Static Resources”, on page 341</td>
</tr>
</tbody>
</table>
5.14.1.3 Static Custom Resources

A static resource is one which is under the control of PBS. Any changes to the value are performed by PBS or by the administrator. Static custom resources are defined ahead of time, at the server, queues or vnodes. Static custom resources can be local or global.

5.14.1.3.i Global Static Custom Resources

Global static custom resources are defined in PBS_HOME/server_priv/resource-def. Global static custom resource values at vnode, queue and server are set via qmgr, by setting resources_available.<custom resource name> = <value>. These resources are available at the server, queues, or vnodes.

5.14.1.3.ii Local Static Custom Resources

Local static custom resources are defined in PBS_HOME/mom_priv/config, and are available only at the host where they are defined. Note that these resources cannot be set via qmgr or viewed via qstat. It is not recommended to use local static custom resources.

5.14.1.4 Shared Vnode Resources

A shared vnode resource is managed at one vnode, but available to be used by jobs at others. This allows flexible allocation of the resource. See section 5.14.5.3, “Shared Host-level Resources”, on page 339 for information on resources shared across vnodes.

5.14.1.5 Using Custom Resources for Application Licenses

The following table lists application licenses and what kind of custom resource to define for them. See section 5.14.7, “Supplying Application Licenses”, on page 346 for specific instructions on configuring each type of license and examples of configuring custom resources for application licenses.

<table>
<thead>
<tr>
<th>Floating or Node-locked</th>
<th>Unit Being Licensed</th>
<th>How License is Managed</th>
<th>Level</th>
<th>Resource Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating (site-wide)</td>
<td>Token</td>
<td>External license manager</td>
<td>Server</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Floating (site-wide)</td>
<td>Token</td>
<td>PBS</td>
<td>Server</td>
<td>Static</td>
</tr>
</tbody>
</table>
You can configure a custom resource to report how much scratch space is available on machines. Jobs requiring scratch space can then be scheduled onto machines which have enough. This requires dynamic host-level resources. See section 5.14.6, “Using Scratch Space”, on page 345 and section 5.14.5.1, “Dynamic Host-level Resources”, on page 337.

### 5.14.2 Defining New Custom Resources

When you define a new custom resource to be used by jobs, you must do the following:

1. Define the resource to the server in the server’s `resource.def` file
2. Put the resource in the “resources:” line in `/PBS_HOME/sched_priv/sched_config`. If the resource is a host-level boolean, you do not need to add it here.
3. Set the resource, either via `qmgr` or by adding it to the correct configuration line
4. If the resource is dynamic, it must be added to the correct line in the scheduler’s configuration file:
   - If it’s a host-level dynamic resource, it must be added to the `mom_resources` line
   - If it’s a server-level resource, it must be added to the `server_dyn_res` line
5. Depending on the type of resource, the server, scheduler and MoMs must be restarted. See section 5.14.3, “Restart Steps for Custom Resources”, on page 332.

### 5.14.2.1 Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

---

**Table 5-5: Custom Resources for Application Licenses**

<table>
<thead>
<tr>
<th>Floating or Node-locked</th>
<th>Unit Being Licensed</th>
<th>How License is Managed</th>
<th>Level</th>
<th>Resource Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-locked</td>
<td>Host</td>
<td>PBS</td>
<td>Host</td>
<td>Static</td>
</tr>
<tr>
<td>Node-locked</td>
<td>CPU</td>
<td>PBS</td>
<td>Host</td>
<td>Static</td>
</tr>
<tr>
<td>Node-locked</td>
<td>Instance of Application</td>
<td>PBS</td>
<td>Host</td>
<td>Static</td>
</tr>
</tbody>
</table>
5.14.2.2 Dynamic Resource Scripts/Programs

You create the script or program that PBS uses to query the external source. The external source can be a license manager or a command, as when you use the `df` command to find the amount of available disk space. If the script is for a server-level dynamic resource, it is placed on the server. If it is for a host-level resource, it is placed on the host(s) where it will be used.

5.14.2.2.i Requirements for Scripts/Programs

- The script must be available to the scheduler, which runs the script
- If you have set up peer scheduling, make sure that the script is available to any scheduler that must run it
- The script must return its output via `stdout`, and the output must be in a single line ending with a newline
- In Windows, if you use Notepad to create the script, be sure to explicitly put a newline at the end of the last line, otherwise none will appear, causing PBS to be unable to properly parse the file

5.14.2.3 Defining and Setting Static and Dynamic Custom Resources

The following table lists the differences in defining and setting static and dynamic custom resources at the Server, Queue and host level.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Server-level</th>
<th>Queue-level</th>
<th>Host-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>Set via <code>qmgr</code></td>
<td>Set via <code>qmgr</code></td>
<td>Set via <code>qmgr</code></td>
</tr>
<tr>
<td>dynamic</td>
<td>Add to <code>server_dyn_res</code> line in PBS_HOME/sched_priv/sched_config</td>
<td>Cannot be used.</td>
<td>Add to MoM config file PBS_HOME/mom_priv/config and mom_resources line in PBS_HOME/sched_priv/sched_config</td>
</tr>
</tbody>
</table>
5.14.2.4 The resourcedef File

Global custom resources are defined in PBS_HOME/server_priv/resourcedef. The format of each line in PBS_HOME/server_priv/resourcedef is:

<resource name> [type=<resource type>] [flag=<resource flag>]

<resource name> is any string made up of alphanumeric characters, where the first character is alphabetic. Resource names must start with an alphabetic character and can contain alphanumeric, underscore ("_"), and dash ("-"), characters: [a-zA-Z][a-zA-Z0-9_]*. The length of each line in PBS_HOME/server_priv/resourcedef file should not be more than 254 characters. There is no limit to the number of custom resources that can be defined.

<resource type> is the type of the resource value, which can be one of the following keywords:

- Boolean
- long
- string
- string_array
- size
- float

You cannot create a custom resource of type "time" or "duration". For these resources, use "long".

The default for <resource type> is "long".

The format of custom Boolean, size, string or string_array resources must be the same as built-in resources.

<resource flag> is zero or more resource accumulation or resource permission flags. See the following sections.

See "Resource Data Types" on page 297 of the PBS Professional Reference Guide for a description of each resource type.

You must restart the server after defining resources in the resourcedef file. See section 5.14.3.1, "Restarting the Server", on page 332.
5.14.2.5 Resource Accumulation Flags

When you define a custom resource, you can specify whether it is server-level or host-level, and whether it is consumable or not. This is done by setting resource accumulation flags in the resource definition in `PBS_HOME/server_priv/resourcedef`. A consumable resource is tracked, or accumulated, in the server, queue or vnode `resources_assigned` attribute. The resource accumulation flags determine where the value of `resources_assigned.<resource>` is incremented.

5.14.2.5.i Allowable Values for Resource Accumulation Flags

The value of `<resource flags>` is a concatenation of one or more of the following letters:

- (none of h, n, f, or q)
  Indicates a queue-level or server-level resource that is not consumable.

- h
  Indicates a host-level resource. Used alone, means that the resource is not consumable. Required for any resource that will be used inside a select statement. This flag selects hardware. This flag indicates that the resource must be requested inside of a select statement.
  Example: for a Boolean resource named "green":
  ```
  green type=boolean flag=h
  ```

- n
  The amount is consumable at the host level, for all vnodes assigned to the job. Must be consumable or time-based. Cannot be used with Boolean or string resources. The “h” flag must also be used.
  This flag specifies that the resource is accumulated at the vnode level, meaning that the value of `resources_assigned.<resource>` is incremented at relevant vnodes when a job is allocated this resource or when a reservation requesting this resource on this vnode starts.
  This flag is not used with dynamic consumable resources. The scheduler will not oversubscribe dynamic consumable resources.

- f
  The amount is consumable at the host level for only the first vnode allocated to the job (vnode with first task.) Must be consumable or time-based. Cannot be used with Boolean or string resources. The “h” flag must also be used.
  This flag specifies that the resource is accumulated at the first vnode, meaning that the value of `resources_assigned.<resource>` is incremented only at the first vnode when a job is allocated this resource or when a reservation requesting this resource on this vnode starts.
q

The amount is consumable at the Queue and Server level. When a job is assigned one unit of a resource with this flag, the `resources_assigned.<resource>` attribute at the server and any queue is incremented by one. Must be consumable or time-based.

This flag specifies that the resource is accumulated at the queue and server level, meaning that the value of `resources_assigned.<resource>` is incremented at each queue and at the server when a job is allocated this resource. When a reservation starts, allocated resources are added to the server’s `resources_assigned` attribute.

This flag is not used with dynamic consumable resources. The scheduler will not oversubscribe dynamic consumable resources.

5.14.2.5.ii When to Use Accumulation Flags

The following table shows when to use accumulation flags.

Table 5-7: When to Use Accumulation Flags

<table>
<thead>
<tr>
<th>Resource Category</th>
<th>Server</th>
<th>Queue</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static, consumable</td>
<td>flags = q</td>
<td>flags = q</td>
<td>flags = nh or fh</td>
</tr>
<tr>
<td>Static, not consum-</td>
<td>flags = (none of h, n, q or f)</td>
<td>flags = (none of h, n, q or f)</td>
<td>flags = h</td>
</tr>
<tr>
<td>able</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic</td>
<td>server_dyn_res line in sched_config, flags = (none of h, n, q or f)</td>
<td>(cannot be used)</td>
<td>MoM config and mom_resources line in sched_config, flags = h</td>
</tr>
</tbody>
</table>

5.14.2.5.iii Example of Resource Accumulation Flags

When defining a static consumable host-level resource, such as a node-locked license, you would use the “n” and “h” flags.

When defining a dynamic resource such as a floating license, you would use no flags.

5.14.2.5.iv Resource Accumulation Flag Restrictions and Caveats

- Numeric dynamic resources cannot have the q or n flags set. This would cause these
resources to be underused. These resources are tracked automatically by the scheduler.

5.14.2.6 Resource Permission Flags

When you define a custom resource, you can specify whether unprivileged users have permission to view or request the resource, and whether users can qalter a request for that resource. This is done by setting two resource permission flags in the resource definition in $PBS_HOME/server_priv/resourcedef.

5.14.2.6.i Allowable Values for Resource Permission Flags

i

“Invisible”. Users cannot view or request the resource. Users cannot qalter a resource request for this resource.

r

“Read only”. Users can view the resource, but cannot request it or qalter a resource request for this resource.

(neither i nor r)

Users can view and request the resource, and qalter a resource request for this resource.

5.14.2.6.ii Effect of Resource Permission Flags

• PBS Operators and Managers can view and request a resource, and qalter a resource request for that resource, regardless of the i and r flags.
• Users, operators and managers cannot submit a job which requests a restricted resource. Any job requesting a restricted resource will be rejected. If a manager needs to run a job which has a restricted resource with a different value from the default value, the manager must submit the job without requesting the resource, then qalter the resource value.
• While users cannot request these resources, their jobs can inherit default resources from resources_default.<resource> and default_chunk.<resource>.

If a user tries to request a resource or modify a resource request which has a resource permission flag, they will get an error message from the command and the request will be
rejected. For example, if they try to `qalter` a job’s resource request, they will see an error message similar to the following:

```
qalter: Cannot set attribute, read only or insufficient permission
Resource_List.hps 173.mars
```

Example resourcedef file:

```
W_prio  type=long  flag=i
B_prio  type=long  flag=r
P_prio  type=long  flag=i
```

5.14.2.6.iii Resource Permission Flag Restrictions and Caveats

- You can specify only one of the `i` or `r` flags per resource. If both are specified, the resource is treated as if only the `i` flag were specified, and an error message is logged at the default log event class and printed to standard error.

- Resources assigned from the `default_qsub_arguments` server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag whether that resource was requested by the user or came from `default_qsub_arguments`.

- The behavior of several command-line interfaces is dependent on resource permission flags. These interfaces are those which view or request resources or modify resource requests:

  - `pbsnodes`
    - Users cannot view restricted host-level custom resources.
  
  - `pbs_rstat`
    - Users cannot view restricted reservation resources.
  
  - `pbs_rsub`
    - Users cannot request restricted custom resources for reservations.
  
  - `qalter`
    - Users cannot alter a restricted resource.
  
  - `qmgr`
    - Users cannot print or list a restricted resource.
  
  - `qselect`
    - Users cannot specify restricted resources via `-l Resource_List`.

  - `qsub`
    - Users cannot request a restricted resource.

  - `qstat`
    - Users cannot view a restricted resource.
5.14.2.7 Example of Defining Each Type of Custom Resource

In this example, we add five custom resources: a static and a dynamic host-level resource, a static and a dynamic server-level resource, and a static queue-level resource.

1. The resource must be defined to the server, with appropriate flags set:
   Add resource to PBS_HOME/server_priv/resourcedef:
   ```
   staticserverresource   type=long flag=q
   statichostresource     type=long flag=nh
   dynamicserverresource  type=long
   dynamichostresource    type=long flag=h
   staticqueueresource    type=long flag=q
   ```


3. The resource must be added to the scheduler’s list of resources:
   Add resource to “resources:” line in PBS_HOME/sched_priv/sched_config:
   ```
   resources:  "[...], staticserverresource, statichostresource,
              dynamicserverresource, dynamichostresource, staticqueueresource"
   ```
   Host-level Boolean resources do not need to be added to the “resources:” line.

4. HUP the scheduler:
   ```
   kill -HUP <scheduler PID>
   ```

5. If the resource is static, use qmgr to set it at the host, queue or server level:
   ```
   Qmgr: set node Host1 resources_available.statichostresource=1
   Qmgr: set queue Queue1 resources_available.staticqueueresource=1
   Qmgr: set server resources_available.staticserverresource=1
   ```
   See “qmgr” on page 150 of the PBS Professional Reference Guide.

6. If the resource is dynamic:
   a. If it’s a host-level resource, add it to the “mom_resources” line in PBS_HOME/
sched_priv/sched_config:
  mom_resources: “dynamichostresource”

b. Add it to the MoM config file PBS_HOME/mom_priv/config:
UNIX or Windows:
  dynamichostresource !path-to-command
Windows, spaces in path:
  dynamichostresource !“path-to-command”

c. If it’s a server-level resource, add it to the “server_dyn_res” line in PBS_HOME/sched_priv/sched_config:
UNIX:
  server_dyn_res: “dynamicserverresource !path-to-command”
Windows, no spaces in path:
  server_dyn_res: ‘dynamicserverresource !path-to-command’
or:
  server_dyn_res: “dynamicserverresource !path-to-command”
Windows, spaces in path:
  server_dyn_res: ‘dynamicserverresource !“path-to-command including spaces”’

5.14.2.8 Custom Resource Values

Allowable values for float and long resources are the same as for built-in resources. If a string resource value contains spaces or shell metacharacters, enclose the string in quotes, or otherwise escape the space and metacharacters. Be sure to use the correct quotes for your shell and the behavior you want. If the string resource value contains commas, the string must be enclosed in an additional set of quotes so that the command (e.g. qsub, qalter) will parse it correctly. If the string resource value contains quotes, plus signs, equal signs, colons or parentheses, the string resource value must be enclosed in yet another set of additional quotes.
5.14.3 Restart Steps for Custom Resources

In order to have new custom resources recognized by PBS, you must restart or reinitialize any PBS daemon whose files were changed.

5.14.3.1 Restarting the Server

The server must always be restarted for a new custom resource to be recognized.

5.14.3.1.i Restarting the Server on UNIX/Linux

```
qterm -t quick
PBS_EXEC/sbin/pbs_server
```

5.14.3.1.ii Restarting the Server on Windows

```
Admin> qterm -t quick
Admin> net start pbs_server
```

5.14.3.1.iii Restarting the Server with Failover Configured

Using `qterm -t quick` leaves the secondary server running; it will become active. If you have configured failover, see section 9.2.7.1, “Stopping Servers”, on page 708 and section 9.2.7.2, “Starting Servers”, on page 709.

5.14.3.2 Restarting or Reinitializing MoM

On UNIX/Linux, whether the MoM must be restarted or reinitialized depends on which MoM configuration file has been changed.

- If only the Version 1 MoM configuration file was changed, you only need to HUP the MoM.
- If you used the `pbs_mom -s insert` command to add to or change anything in the Version 2 MoM config file, you can HUP the MoM.
- If you used the `pbs_mom -s insert` command to remove anything from the Version 2 MoM config file, you must restart the MoM.

On Windows, you must restart MoM when any MoM configuration file has been changed.
5.14.3.2.i  Reinitializing MoM on UNIX/Linux

1. Use the ps command to determine MoM’s process ID. Note that ps arguments vary among UNIX systems, thus “-ef” may need to be replaced by “-aux”.
   
   ```
   ps –ef | grep pbs_mom
   ```

2. HUP MoM using the kill command, with MoM’s PID as an argument:
   
   ```
   kill -HUP <MoM PID>
   ```

See “pbs_mom” on page 58 of the PBS Professional Reference Guide.

5.14.3.2.ii  Restarting MoM on UNIX/Linux

1. Use the ps command to determine MoM’s process ID. Note that ps arguments vary among UNIX systems, thus “-ef” may need to be replaced by “-aux”.
   
   ```
   ps –ef | grep pbs_mom
   ```

2. Terminate MoM using the kill command, with MoM’s PID as an argument. The syntax will vary depending on your system:
   
   ```
   kill -INT <MoM PID>
   ```
   
   or
   
   ```
   kill -s INT <MoM PID>
   ```

3. Restart MoM, allowing running jobs to continue running through the restart. If your custom resource query script/program takes longer than the default ten seconds, you can change the alarm timeout via the -a alarm command line start option:
   
   ```
   PBS_EXEC/sbin/pbs_mom -p [ -a timeout]
   ```

See “pbs_mom” on page 58 of the PBS Professional Reference Guide.

5.14.3.2.iii  Restarting MoM on Windows

If your custom resource query script/program takes longer than the default ten seconds, you can change the alarm timeout via the -a alarm command line start option.

```
Admin> net stop pbs_mom
Admin> net start pbs_mom
```

See “Startup Options to PBS Services” on page 188 in the PBS Professional Installation & Upgrade Guide.

5.14.3.3  Restarting the Scheduler

You must restart the scheduler if you added the new custom resource to the resources: line in PBS_HOME/sched_priv/sched_config.
5.14.3.3.i Reinitializing the Scheduler on UNIX/Linux

```
ps -ef | grep pbs_sched
kill -HUP <Scheduler PID>
```

5.14.3.3.ii Restarting the Scheduler on Windows

```
Admin> net stop pbs_sched
Admin> net start pbs_sched
```

5.14.4 Configuring Server-level Resources

5.14.4.1 Dynamic Server-level Resources

The availability of a dynamic server-level resource is determined by running a script or program specified in the `server_dyn_res` line of `PBS_HOME/sched_priv/sched_config`. The value for `resources_available.<resource>` is updated at each scheduling cycle with the value returned by the script. This script is run at the host where the scheduler runs, once per scheduling cycle. The script must return the value via `stdout` in a single line ending with a newline.

The scheduler tracks how much of each numeric dynamic server-level custom resource has been assigned to jobs, and will not overcommit these resources.

The format of a dynamic server-level resource query is a shell escape:

`server_dyn_res: "<resource name> !<path to command>"`

In this query, 

`<resource name>` is identical to the name in the `resourcedef` file.

`<path to command>` is typically the full path to the script or program that performs the query in order to determine the status and/or availability of the new resource you have added.

The scheduler runs the query and waits for it to finish.

Dynamic server-level resources are usually used for site-wide externally-managed floating licenses.

Server dynamic resource values are never visible in `qstat`, and have no `resources_available.<resource>` representation anywhere in PBS. If a job has requested a server dynamic resource, then the requested value shows up in the output of `qstat`.
5.14.4.1.i Example of Configuring Dynamic Server-level Resource

For a site-wide externally-managed floating license you will need two resources: one to represent the licenses themselves, and one to mark the vnodes on which the application can be run. The first is a server-level dynamic resource and the second is a host-level Boolean, set on the vnodes to send jobs requiring that license to those vnodes.

These are the steps for configuring a dynamic server-level resource for a site-wide externally-managed floating license. If this license could be used on all vnodes, the Boolean resource would not be necessary.

1. Define the resources, for example floatlicense and CanRun, in the server resource definition file PBS_HOME/server_priv/resourcedef:
   
   floatlicense type=long
   CanRun type=boolean flag=h

2. Write a script, for example serverdyn.pl, that returns the available amount of the resource via stdout, and place it on the server’s host. For example, it could be placed in /usr/local/bin/serverdyn.pl


4. Configure the scheduler to use the script by adding the resource and the path to the script in the server_dyn_res line of PBS_HOME/sched_priv/sched_config:

   UNIX:
   server_dyn_res: “floatlicense !/usr/local/bin/serverdyn.pl”

   Windows:
   server_dyn_res: ‘floatlicense !“C:\Program Files\PBS Pro\serverdyn.pl”’

5. Add the new dynamic resource to the resources: line in PBS_HOME/sched_priv/sched_config:

   resources: “ncpus, mem, arch, […], floatlicense”


7. Set the Boolean resource on the vnodes where the floating licenses can be run. Here we designate vnode1 and vnode2 as the vnodes that can run the application:

   Qmgr: active node vnode1,vnode2
   Qmgr: set node resources_available.CanRun=True
Chapter 5  PBS Resources

To request this resource, the job's resource request would include:

-1 floatlicense=<number of licenses or tokens required>
-1 select=1:ncpus=N:CanRun=1

5.14.4.2  Static Server-level Resources

Static server-level resources are used for resources like floating licenses that PBS will manage. PBS keeps track of the number of available licenses instead of querying an external license manager.

5.14.4.2.i  Example of Configuring Static Server-level Resource

These are the steps for configuring a static server-level resource:

1. Define the resource, for example sitelicense, in the server resource definition file
   PBS_HOME/server_priv/resourcedef:
   sitelicense type=long flag=q


3. Use the qmgr command to set the value of the resource on the server:

   Qmgr: set server resources_available.sitelicense=<number of licenses>

4. Add the new resource to the resources: line in PBS_HOME/sched_priv/
   sched_config.

   resources: “ncpus, mem , arch, [...], sitelicense”


5.14.5  Configuring Host-level Custom Resources

Host-level custom resources can be static and consumable, static and not consumable, or dynamic. Dynamic host-level resources are used for things like scratch space.
5.14.5.1 Dynamic Host-level Resources

For dynamic host-level custom resources, the scheduler sends a resource query to each MoM to get the current availability for the resource, and uses that value for scheduling. If the MoM returns a value, this value replaces the resources_available value reported by the Server. If the MoM returns no value, the value from the Server is kept. If neither specifies a value, the Scheduler sets the resource value to 0.

The available amount of the resource is determined by running a script or program which returns the amount via stdout. This script or program is specified in the mom_resources line in PBS_HOME/sched_priv/sched_config.

The script is run once per scheduling cycle. For a multi-vnode machine, the script is run for the natural vnode. The resource is shared among the MoM’s vnodes.

The scheduler tracks how much of each numeric dynamic server-level custom resource has been assigned to jobs, and will not overcommit these resources.

The format of a dynamic host-level resource query is a shell escape:

<resource name> !<path to command>

In this query,

<resource name> is identical to the name in the resourcedef file.

<path to command> is typically the full path to the script or program that performs the query in order to determine the status and/or availability of the new resource you have added.

MoM starts the query and waits for output. The default amount of time that MoM waits is 10 seconds; this period can be set via the -a alarm_timeout command line option to pbs_mom. See section 5.14.3.2, “Restarting or Reinitializing MoM”, on page 332 and “Startup Options to PBS Services” on page 188 in the PBS Professional Installation & Upgrade Guide. If the timeout is exceeded and the shell escape process has not finished, a log message, “resource read alarm” is written to the MoM’s log file. The process is given another alarm period to finish and if it does not, another log message is written. The user’s job may not run.

An example of a dynamic host-level resource is scratch space on the execution host.

Host dynamic resource values are never visible in qstat, and have no resources_available.<resource> representation anywhere in PBS.
Example of Configuring Dynamic Host-level Resource

In this example, we configure a custom resource to track host-level scratch space. The resource is called *dynscratch*. These are the steps for configuring a dynamic host-level resource:

1. Write a script, for example `hostdyn.pl`, that returns the available amount of the resource via `stdout`. The script must return the value in a single line, ending with a newline. Place the script on each host where it will be used. For example, it could be placed in `/usr/local/bin/hostdyn.pl`.

2. Configure each MoM to use the script by adding the resource and the path to the script in `PBS_HOME/mom_priv/config`:
   
   **UNIX:**
   ```bash
   dynscratch !/usr/local/bin/hostdyn.pl
   ```
   
   **Windows:**
   ```bash
   dynscratch !“C:\Program Files\PBS Pro\hostdyn.pl”
   ```

3. Reinitialize the MoMs. See [section 5.14.3.2, “Restarting or Reinitializing MoM”, on page 332](#).

4. Define the resource, for example `dynscratch`, in the server resource definition file `PBS_HOME/server_priv/resourcedef`:
   ```bash
   dynscratch type=size flag=h
   ```

5. Restart the server. See [section 5.14.3.1, “Restarting the Server”, on page 332](#).

6. You may optionally specify any limits on that resource via `qmgr`, such as the maximum amount available, or the maximum that a single user can request. For example:
   ```bash
   qmgr: set server resources_max.scratchspace=1gb
   ```

7. Add the new resource to the `resources:` line in `PBS_HOME/sched_priv/sched_config`:
   ```bash
   resources: “ncpus, mem, arch, [...], dynscratch”
   ```

8. Add the new resource to the `mom_resources:` line in `PBS_HOME/sched_priv/sched_config`. Create the line if necessary:
   ```bash
   mom_resources: “dynscratch”
   ```

9. Restart the scheduler. See [section 5.14.3.3, “Restarting the Scheduler”, on page 333](#).

To request this resource, the resource request would include:

```
-l select=1:ncpus=N:dynscratch=10MB
```
5.14.5.2 Static Host-level Resources

Use static host-level resources for things that are managed by PBS and available at the host level, such as GPUs.

5.14.5.2.i Example of Configuring Static Host-level Resource

In this example, we configure a consumable host-level resource to track GPUs. These are the steps for configuring a static host-level resource:

1. Define the resource, for example `ngpus`, in the server resource definition file
   
   ```
   PBS_HOME/server_priv/resourcedef:
   ngpus type=long flag=nh
   ```


3. Use the `qmgr` command to set the value of the resource on the host:
   
   ```
   qmgr: set node Host1 ngpus=<number of GPUs>
   ```

4. Add the new resource to the `resources` line in `PBS_HOME/sched_priv/sched_config`.
   
   ```
   resources: “ncpus, mem, arch, [...], ngpus”
   ```

5. Restart the scheduler. See section 5.14.3.3, “Restarting the Scheduler”, on page 333.

6. If the GPU host is a multi-vnode machine, you may want to define which GPUs belong in which vnodes. In this case, do the following:


5.14.5.3 Shared Host-level Resources

Two or more vnodes can share the use of a resource. The resource is managed at one vnode, but available for use at other vnodes. The MoM manages the sharing of the resource, allocating only the available amount to jobs. For example, if you want jobs at two separate vnodes...
to be able to use the same 4GB of memory, you can make the memory be a shared resource. This way, if a job at one vnode uses all 4GB, no other jobs can use it, but if one job at one vnode uses 2GB, other jobs at either vnode can use up to 2GB.

5.14.5.3.i  **Shared Resource Terminology**

**Borrowing vnode**
The vnode where a shared vnode resource is available, but not managed.

**Indirect resource**
A shared vnode resource at vnode(s) where the resource is not defined, but which share the resource.

**Managing vnode**
The vnode where a shared vnode resource is defined, and which manages the resource.

**Shared resource**
A vnode resource defined at managed at one vnode, but available for use at others.

5.14.5.3.ii  **Configuring Shared Host-level Resources**

The resource to be shared is defined as usual at one vnode. This is the managing vnode for that resource. For example, to make memory be managed at Vnode1:

    Qmgr: set node Vnode1 mem = 4gb

At vnodes which will use the same resource, the resource is defined to be indirect. For example, to make memory be shared and borrowed at Vnode2:

    Qmgr: set node Vnode2 mem = @Vnode1

5.14.5.3.iii  **Shared Dynamic Host-level Resources**

Vnode-level dynamic resources, meaning those listed in the `mom_resources:` line in `PBS_HOME/sched_priv/sched_config`, are shared resources.

5.14.5.3.iv  **Shared Static Host-level Resources**

You can define a static host-level resource to be shared between vnodes. The resource is not shared if you set it to a value at each vnode.
### 5.14.5.3.v Configuring Shared Static Resources

1. If the resource to be shared is a custom resource, you must define the resource in `PBS_HOME/server_priv/resourcedef` before setting its value:
   ```
   <resource name> type=<resource type> [flag = <flags>]
   ```


3. Set the resource on the managing vnode:
   ```
   To set a static value via qmgr:
   Qmgr: s n managing_vnode resources_available.<resource> =<value>
   To set a static value, in MoM Version 2 configuration file:
   managing_vnode:<resource>=<value>
   ```

4. Next, set the resource on the borrowing vnode:
   ```
   To set a shared resource on a borrowing vnode via qmgr:
   Qmgr: s n borrowing_vnode resources_available.<resource>=@managing_vnode
   To set a shared resource in MoM Version 2 configuration file:
   borrowing_vnode:<resource>=@managing_vnode
   ```

5. HUP the MoMs involved; see section 5.14.3.2, “Restarting or Reinitializing MoM”, on page 332.

Example 5-10: To make a static host-level license `dyna-license` on hostA be managed by the natural vnode at hostA and indirect at vnodes hostA0 and hostA1:

```qmgr
Qmgr: set node hostA resources_available.dyna-license=4
Qmgr: set node hostA0 resources_available.dyna-license=@hostA
Qmgr: set node hostA1 resources_available.dyna-license=@hostA
```

### 5.14.5.3.vi Restrictions on Shared Host-level Resources

- If your vnodes represent physical units such as blades, sharing resources like `ncpus` across vnodes may not make sense.

- If you want to make a resource shared across vnodes, remember that you do not want to schedule jobs on the natural vnode. To avoid this, the following resources should not be explicitly set on the natural vnode:
  ```
  ncpus
  mem
  vmem
  ```
5.14.5.3.vii  Defining Shared and Non-shared Resources for the Altix

On an Altix where you are running `pbs_mom.cpuset`, you can manage the resources at each vnode. For dynamic host-level resources, the resource is shared across all the vnodes on the machine, and MoM manages the sharing. For static host-level resources, you can either
define the resource as shared or not. Shared resources are usually set on the natural vnode and then made indirect at any other vnodes on which you want the resource available. For resources that are not shared, you can set the value at each vnode.

Example 5-11: To set the resource `string_res` to `round` on the natural vnode of `altix03` and make it indirect at `altix03[0]` and `altix03[1]`:

```
Qmgr: set node altix03 resources_available.string_res=round
Qmgr: s n altix03[0] resources_available.string_res=@altix03
Qmgr: s n altix03[1] resources_available.string_res=@altix03
pbsnodes -va
altix03
...
string_res=round
...
altix03[0]
...
string_res=@altix03
...
altix03[1]
...
string_res=@altix03
...
```

If you had set the resource `string_res` individually on `altix03[0]` and `altix03[1]`:

```
Qmgr: s n altix03[0] resources_available.string_res=round
Qmgr: s n altix03[1] resources_available.string_res=square
pbsnodes -va
altix03
...
<--------string_res not set on natural vnode
...
altix03[0]
...
string_res=round
...
altix03[1]
...
string_res=square
5.14.5.3.viii  Shared Resource Restrictions for Multi-vnode Machines

- Do not set the values for `mem`, `vmem` or `ncpus` on the natural vnode. If any of these resources has been explicitly set to a non-zero value on the natural vnode, set `resources_available.ncpus`, `resources_available.mem` and `resources_available.vmem` to zero on each natural vnode. See section 3.5.2.3, “Configuring Machines with Cpusets”, on page 52.

- On the natural vnode, all values for `resources_available.<resource>` should be zero (0), unless the resource is being shared among other vnodes via indirection.
5.14.6 Using Scratch Space

5.14.6.1 Dynamic Server-level (Shared) Scratch Space

If you have scratch space set up so that it’s available to all execution hosts, you can use a server-level custom dynamic resource to track it. The following are the steps for configuring a dynamic server-level resource called `globalscratch` to track globally available scratch space:

1. Define the resource in the server resource definition file `PBS_HOME/server_priv/resourcedef`:
   ```
globalscratch type=long
   ```
2. Write a script, for example `serverdynscratch.pl`, that returns the available amount of the resource via `stdout`, and place it on the server’s host. For example, it could be placed in `/usr/local/bin/serverdynscratch.pl`
4. Configure the scheduler to use the script by adding the resource and the path to the script in the `server_dyn_res` line of `PBS_HOME/sched_priv/sched_config`:
   UNIX:
   ```
   server_dyn_res: "globalscratch !/usr/local/bin/serverdynscratch.pl"
   ```
   Windows:
   ```
   server_dyn_res: 'globalscratch !"C:\Program Files\PBS\Pro\serverdynscratch.pl"'
   ```
5. Add the new dynamic resource to the `resources:` line in `PBS_HOME/sched_priv/sched_config`:
   ```
   resources: "ncpus, mem , arch, […], globalscratch"
   ```

To request this resource, the job’s resource request would include:
```
-l globalscratch=<space required>
```

5.14.6.2 Dynamic Host-level Scratch Space

Say you have jobs that require a large amount of scratch disk space during their execution. To ensure that sufficient space is available during job startup, create a custom dynamic resource so that jobs can request scratch space. To create this resource, take the steps outlined in section 5.14.5.1.i, “Example of Configuring Dynamic Host-level Resource”, on page 338.
5.14.6.3 Static Server-level Scratch Space

If you want to prevent jobs from stepping on each others’ scratch space, you can define additional vnodes that are used only to allocate scratch devices, with one vnode per scratch device. Set the sharing attribute on each scratch vnode to force_excl, so that only one job can request each scratch device. To set the sharing attribute, follow the rules in section 3.5.2, “Choosing Configuration Method”, on page 52. For example, the scratch devices are /scratch1, /scratch2, /scratch3, etc. On each scratch device, set resources as follows:

```
resources_available.ncpus = 0
resources_available.mem = 0
resources_available.scratch = 1
sharing = force_excl
```

Jobs then request one additional chunk to represent the scratch device, for example:

```
-l 16:ncpus=1+1:scratch=1
```

If a job needs to request a specific scratch device, for example /scratch2, that can be done by additionally asking for the vnode explicitly:

```
:vnode=scratch2
```

5.14.6.4 Static Host-level Scratch Space

If the scratch areas are not mounted on all execution hosts, you can specify which scratch areas are shared among which subsets of vnodes using indirect resources. See section 5.14.5.3, “Shared Host-level Resources”, on page 339.

5.14.6.5 Caveats for Scratch Space and Jobs

When more than one job uses scratch space, or when a job is suspended, scratch space usage may not be handled correctly. See section 5.9.5, “Dynamic Resource Allocation Caveats”, on page 313 and section 5.9.6, “Period When Resource is Used by Job”, on page 313.

5.14.7 Supplying Application Licenses

5.14.7.1 Types of Licenses

Application licenses may be managed by PBS or by an external license manager. Application licenses may be floating or node-locked, and they may be per-host, where any number of instances can be running on that host, per-CPU, where one license allows one CPU to be used for that application, or per-run, where one license allows one instance of the application to be running. Each kind of license needs a different form of custom resource.
5.14.7.1.i  Externally-managed Licenses
Whenever an application license is managed by an external license manager, you must create a custom dynamic resource for it. This is because PBS has no control over whether these licenses are checked out, and must query the external license manager for the availability of those licenses. PBS does this by executing the script or program that you specify in the dynamic resource. This script returns the amount via `stdout`, in a single line ending with a newline.

5.14.7.1.ii  Preventing Oversubscription of Externally-managed Licenses
Some applications delay the actual license checkout until some time after the application begins execution. Licenses could be oversubscribed when the scheduler queries for available licenses, and gets a result including licenses that essentially belong to a job that is already running but has not yet checked them out. To prevent this, you can create a consumable custom static integer resource, assign it the total number of licenses, and make each job that requests licenses request this resource as well. You can use a hook to accomplish this. Alternatively, if you know the maximum number of jobs that can run using these licenses, you can create a consumable custom static integer resource to track the number of jobs using licenses, and make each job request this resource.

If licenses are also checked out by applications outside of the control of PBS, this technique will not work.

5.14.7.1.iii  PBS-managed Licenses
When an application license is managed by PBS, you can create a custom static resource for it. You set the total number of licenses using `qmgr`, and PBS will internally keep track of the number of licenses available.

Use static host-level resources for node-locked application licenses managed by PBS, where PBS is in full control of the licenses. These resources are static because PBS tracks them internally, and host-level because they are tracked at the host.

5.14.7.2  License Units and Features
Different licenses use different license units to track whether an application is allowed to run. Some licenses track the number of CPUs an application is allowed to run on. Some licenses use tokens, requiring that a certain number of tokens be available in order to run. Some licenses require a certain number of features to run the application.

When using units, after you have defined the license resource called `license_name` to the server, be sure to set `resources_available.license_name` to the correct number of units.
Before starting you should have answers to the following questions:

- How many units of a feature does the application require?
- How many features are required to execute the application?
- How do I query the license manager to obtain the available licenses of particular features?

With these questions answered you can begin configuring PBS Professional to query the license manager servers for the availability of application licenses. Think of a license manager feature as a resource. Therefore, you should associate a resource with each feature.

5.14.7.3 Server-level (Floating) Licenses

5.14.7.3.i Example of Floating, Externally-managed License

Here is an example of setting up floating licenses that are managed by an external license server.

For this example, we have a 6-host complex, with one CPU per host. The hosts are numbered 1 through 6. On this complex we have one licensed application which uses floating licenses from an external license manager. Furthermore we want to limit use of the application only to specific hosts. The table below shows the application, the number of licenses, the hosts on which the licenses should be used, and a description of the type of license used by the application.

<table>
<thead>
<tr>
<th>Application</th>
<th>Licenses</th>
<th>Hosts</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppF</td>
<td>4</td>
<td>3-6</td>
<td>Uses licenses from an externally managed pool</td>
</tr>
</tbody>
</table>

For the floating licenses, we will use three resources. One is a dynamic server resource for the licenses themselves. One is a global server-level integer to prevent oversubscription. The last is a Boolean resource used to indicate that the floating license can be used on a given host.
Server Configuration

1. Define the new resources in the Server’s `resourcedef` file. Create a new file if one does not already exist by adding the resource names, type, and flag(s).

   ```bash
cd $PBS_HOME/server_priv/
[edit] resourcedef
```

   Example `resourcedef` file with new resources added:
   ```
   AppF type=long
   AppFcount type=long flag=q
   runsAppF type=boolean flag=h
   ```


Host Configuration

3. Set the Boolean resource on the hosts where the floating licenses can be used.

   ```bash
   Qmgr: active node host3,host4,host5,host6
   Qmgr: set node resources_available.runsAppF = True
   ```

Scheduler Configuration

4. Edit the Scheduler configuration file:

   ```bash
cd $PBS_HOME/sched_priv/
[edit] sched_config
```

5. Append the new resource names to the resources: line:

   ```
   resources: “ncpus, mem, arch, host, […], AppF, AppFcount, runsAppF”
   ```

6. Edit the server_dyn_res: line:

   UNIX:
   ```
   server_dyn_res: “AppF !/local/flex_AppF”
   ```

   Windows:
   ```
   server_dyn_res: ‘AppF !“C:\Program Files\PBS Pro\flex_AppF”’
   ```


You can write a hook that examines the number of AppF licenses requested by each job, and assigns that many AppFcount to the job, or you can ask your users to request AppFcount.

To request a floating license for AppF and a host on which AppF can run:

   ```bash
   qsub -l AppF=1 -l AppFcount=1
   -l select=runsAppF=True
   ```
The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well.

```
host1
host2
host3
    resources_available.runsAppF = True
host4
    resources_available.runsAppF = True
host5
    resources_available.runsAppF = True
host6
    resources_available.runsAppF = True
```

5.14.7.3.ii Example of Floating, Externally-managed License with Features

This is an example of a floating license, managed by an external license manager, where the application requires a certain number of features to run. Floating licenses are treated as server-level dynamic resources. The license server is queried by an administrator-created script. This script returns the value via `stdout` in a single line ending with a newline.

The license script runs on the server’s host once per scheduling cycle and queries the number of available licenses/tokens for each configured application.

When submitting a job, the user’s script, in addition to requesting CPUs, memory, etc., also requests licenses.

When the scheduler looks at all the enqueued jobs, it evaluates the license request alongside the request for physical resources, and if all the resource requirements can be met the job is run. If the job's token requirements cannot be met, then it remains queued.

PBS doesn't actually check out the licenses; the application being run inside the job's session does that. Note that a small number of applications request varying amounts of tokens during a job run.
Our example needs four features to run an application, so we need four custom resources.

1. Write four scripts, one to query the license server for each of your four features. Complexity of the script is entirely site-specific due to the nature of how applications are licensed.

2. Define four non-consumable server-level features in PBS_HOME/server_priv/resourcedef. These features are defined with no flags:
   ```
   feature1   type=long
   feature3   type=long
   feature6   type=long
   feature8   type=long
   ```


4. Add the feature resources to the resources: line in PBS_HOME/sched_priv/sched_config:
   ```
   resources: “ncpus, mem, arch, […], feature1, feature3, feature6, feature8”
   ```

5. Add each feature’s script path to the server_dyn_res: line in PBS_HOME/server_priv/config:
  
   **UNIX:**
   ```
   server_dyn_res: “feature1 !/path/to/script [args]”
   server_dyn_res: “feature3 !/path/to/script [args]”
   server_dyn_res: “feature6 !/path/to/script [args]”
   server_dyn_res: “feature8 !/path/to/script [args]”
   ```

   **Windows:**
   ```
   server_dyn_res: ‘feature1 !”C:\Program Files\PBS Pro\script [args]”’
   server_dyn_res: ‘feature3 !”C:\Program Files\PBS Pro\script [args]”’
   server_dyn_res: ‘feature6 !”C:\Program Files\PBS Pro\script [args]”’
   server_dyn_res: ‘feature8 !”C:\Program Files\PBS Pro\script [args]”’
   ```


**5.14.7.3.iii Example of Floating License Managed by PBS**

Here is an example of configuring custom resources for a floating license that PBS manages. For this you need a server-level static resource to keep track of the number of available licenses. If the application can run only on certain hosts, then you will need a host-level Boolean resource to direct jobs running the application to the correct hosts.
In this example, we have six hosts numbered 1-6, and the application can run on hosts 3, 4, 5 and 6. The resource that will track the licenses is called AppM. The Boolean resource is called RunsAppM.

Server Configuration

1. Define the new resource in the Server's resourcedef file. Create a new file if one does not already exist by adding the resource names, type, and flag(s).

   ```
cd $PBS_HOME/server_priv/
[edit] resourcedef

Example resourcedef file with new resources added:

AppM  type=long  flag=q
runsAppM type=boolean  flag=h
```


3. Set a value for AppM at the Server. Here, we're allowing 8 copies of the application to run at once:

   ```
Qmgr: set server resources_available.AppM=8
```

Host Configuration

4. Set the value of runsAppM on the hosts. Each qmgr directive is typed on a single line:

   ```
Qmgr: active node host3,host4,host5,host6
Qmgr: set node resources_available.runsAppM = True
```

Scheduler Configuration

5. Edit the Scheduler configuration file:

   ```
cd $PBS_HOME/sched_priv/
[edit] sched_config
```

6. Append the new resource name to the resources: line. Note that it is not necessary to add a host-level Boolean resource to this line.

   ```
resources: “ncpus, mem, arch, host, […], AppM, runsAppM”
```


To request both the application and a host that can run AppM:

   ```
qsub -l AppM=1
   -l select=1:runsAppM=1 <jobscript>
```
The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well. Since unset Boolean resources are the equivalent of `False`, you do not need to explicitly set them to `False` on the other hosts. Unset Boolean resources will not be printed.

```
host1
host2
host3
    resources_available.runsAppM = True
host4
    resources_available.runsAppM = True
host5
    resources_available.runsAppM = True
host5
    resources_available.runsAppM = True
```

### 5.14.7.4 Host-level (Node-locked) Licenses

#### 5.14.7.4.i Per-host Node-locked Licenses

If you are configuring a custom resource for a per-host node-locked license, where the number of jobs using the license does not matter, use a host-level Boolean resource on the appropriate host. This resource is set to `True`. When users request the license, they can use the following requests:

For a two-CPU job on a single vnode:

```
-1 select=1:ncpus=2:license=1
```

For a multi-vnode job:

```
-1 select=2:ncpus=2:license=1
-1 place=scatter
```

Users can also use "license=True", but this way they do not have to change their scripts.

#### 5.14.7.4.ii Per-CPU Node-locked Licenses

If you are configuring a custom resource for a per-CPU node-locked license, use a host-level consumable resource on the appropriate vnode. This resource is set to the maximum number of CPUs you want used on that vnode. Then when users request the license, they will use the following request:
For a two-CPU, two-license job:

```
-l select=1:ncpus=2:license=2
```

### 5.14.7.4.iii Per-use Node-locked License

If you are configuring a custom resource for a per-use node-locked license, use a host-level consumable resource on the appropriate host. This resource is set to the maximum number of instances of the application allowed on that host. Then when users request the license, they will use:

For a two-CPU job on a single host:

```
-l select=1:ncpus=2:license=1
```

For a multi-vnode job where each chunk needs two CPUs:

```
-l select=2:ncpus=2:license=1
-l place=scatter
```

### 5.14.7.4.iv Example of Per-host Node-locked Licensing

Here is an example of setting up node-locked licenses where one license is required per host, regardless of the number of jobs on that host.

For this example, we have a 6-host complex, with one CPU per host. The hosts are numbered 1 through 6. On this complex we have a licensed application that uses per-host node-locked licenses. We want to limit use of the application only to specific hosts. The table below shows the application, the number of licenses for it, the hosts on which the licenses should be used, and a description of the type of license used by the application.

<table>
<thead>
<tr>
<th>Application</th>
<th>Licenses</th>
<th>Hosts</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppA</td>
<td>1</td>
<td>1-4</td>
<td>uses a local node-locked application license</td>
</tr>
</tbody>
</table>

For the per-host node-locked license, we will use a Boolean host-level resource called `resources_available.runsAppA`. This will be set to `True` on any hosts that should have the license, and will default to `False` on all others. The resource is not consumable so that more than one job can request the license at a time.
PBS Resources

Server Configuration
1. Define the new resource in the Server’s resourcedef file. Create a new file if one does not already exist by adding the resource names, type, and flag(s).
   
   ```
   cd $PBS_HOME/server_priv/
   [edit] resourcedef
   
   Example resourcedef file with new resources added:
   
   runsAppA type=boolean flag=h
   AppA type=long flag=h
   ```


Host Configuration
3. Set the value of runsAppA on the hosts. Each qmgr directive is typed on a single line:
   
   ```
   Qmgr: active node host1,host2,host3,host4
   Qmgr: set node resources_available.runsAppA = True
   ```

Scheduler Configuration
4. Edit the Scheduler configuration file.
   
   ```
   cd $PBS_HOME/sched_priv/
   [edit] sched_config
   ```

5. Append the new resource name to the “resources:” line. Note that it is not necessary to add the host-level Boolean resource to this line.
   
   ```
   resources: “ncpus, mem, arch, […], AppA, runsAppA”
   ```


To request a host with a per-host node-locked license for AppA:

   ```
   qsub -l select=1:runsAppA=1 <jobscript>
   ```
The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well. Since unset Boolean resources are the equivalent of `False`, you do not need to explicitly set them to `False` on the other hosts. Unset Boolean resources will not be printed.

```
host1
  resources_available.runsAppA = True
host2
  resources_available.runsAppA = True
host3
  resources_available.runsAppA = True
host4
  resources_available.runsAppA = True
host5
host6
```

### 5.14.7.4.v Example of Per-use Node-locked Licensing

Here is an example of setting up per-use node-locked licenses. Here, while a job is using one of the licenses, it is not available to any other job.

For this example, we have a 6-host complex, with 4 CPUs per host. The hosts are numbered 1 through 6. On this complex we have a licensed application that uses per-use node-locked licenses. We want to limit use of the application only to specific hosts. The licensed hosts can run two instances each of the application. The table below shows the application, the number of licenses for it, the hosts on which the licenses should be used, and a description of the type of license used by the application.

<table>
<thead>
<tr>
<th>Application</th>
<th>Licenses</th>
<th>Hosts</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppB</td>
<td>2</td>
<td>1-2</td>
<td>Uses a local node-locked application license</td>
</tr>
</tbody>
</table>

For the node-locked license, we will use one static host-level resource called `resources_available.AppB`. This will be set to 2 on any hosts that should have the license, and to 0 on all others. The "nh" flag combination means that it is host-level and it is consumable, so that if a host has 2 licenses, only two jobs can use those licenses on that host at a time.
Server Configuration

1. Define the new resource in the Server’s resourcedef file. Create a new file if one does not already exist by adding the resource names, type, and flag(s).

   `cd $PBS_HOME/server_priv/
   [edit] resourcedef`

   Example resourcedef file with new resources added:
   `AppB type=long flag=nh`

2. Restart the Server. See "Restarting the Server" on page 332.

Host Configuration

3. Set the value of AppB on the hosts to the maximum number of instances allowed. Each qmgr directive is typed on a single line:

   `Qmgr: active node host1,host2`
   `Qmgr: set node resources_available.AppB = 2`
   `Qmgr: active node host3,host4,host5,host6`
   `Qmgr: set node resources_available.AppB = 0`

Scheduler Configuration

4. Edit the Scheduler configuration file.

   `cd $PBS_HOME/sched_priv/
   [edit] sched_config`

5. Append the new resource name to the resources: line:

   `resources: "ncpus, mem, arch, host, [...], AppB"

6. Restart the Scheduler. See "Restarting the Scheduler" on page 333.

To request a host with a node-locked license for AppB, where you’ll run one instance of AppB on two CPUs:

   `qsub -l select=1:ncpus=2:AppB=1`
Chapter 5  

PBS Resources

The example below shows what the host configuration would look like. What is shown is actually truncated output from the pbsnodes -a command. Similar information could be printed via the qmgr -c "print node @default" command as well.

```plaintext
host1
  resources_available.AppB = 2
host2
  resources_available.AppB = 2
host3
  resources_available.AppB = 0
host4
  resources_available.AppB = 0
host5
  resources_available.AppB = 0
host6
  resources_available.AppB = 0
```

5.14.7.4.vi Example of Per-CPU Node-locked Licensing

Here is an example of setting up per-CPU node-locked licenses. Each license is for one CPU, so a job that runs this application and needs two CPUs must request two licenses. While that job is using those two licenses, they are unavailable to other jobs.

For this example, we have a 6-host complex, with 4 CPUs per host. The hosts are numbered 1 through 6. On this complex we have a licensed application that uses per-CPU node-locked licenses. We want to limit use of the application to specific hosts only. The table below shows the application, the number of licenses for it, the hosts on which the licenses should be used, and a description of the type of license used by the application.

<table>
<thead>
<tr>
<th>Application</th>
<th>Licenses</th>
<th>Hosts</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppC</td>
<td>4</td>
<td>3-4</td>
<td>uses a local node-locked application license</td>
</tr>
</tbody>
</table>

For the node-locked license, we will use one static host-level resource called resources_available.AppC. We will provide a license for each CPU on hosts 3 and 4, so this will be set to 4 on any hosts that should have the license, and to 0 on all others. The "nh" flag combination means that it is host-level and it is consumable, so that if a host has 4 licenses, only four CPUs can be used for that application at a time.
Server Configuration

1. Define the new resource in the Server’s `resourcedef` file. Create a new file if one does not already exist by adding the resource names, type, and flag(s).
   
   ```
   cd $PBS_HOME/server_priv/
   [edit] resourcedef
   
   Example resourcedef file with new resources added:
   
   AppC type=long flag=nh
   ```

2. Restart the Server. See "Restarting the Server" on page 332.

Host Configuration

3. Set the value of AppC on the hosts. Each `qmgr` directive is typed on a single line:
   
   ```
   Qmgr: active node host3,host4
   Qmgr: set node resources_available.AppC = 4
   Qmgr: active node host1,host2,host5,host6
   Qmgr: set node resources_available.AppC = 0
   ```

Scheduler Configuration

4. Edit the Scheduler configuration file:
   
   ```
   cd $PBS_HOME/sched_priv/
   [edit] sched_config
   ```

5. Append the new resource name to the `resources:` line:
   
   ```
   resources: “ncpus, mem, arch, host, [...], AppC”
   ```

6. Restart the Scheduler. See "Restarting the Scheduler" on page 333.

To request a host with a node-locked license for AppC, where you’ll run a job using two CPUs:

```
qsub -l select=1:ncpus=2:AppC=2
```
The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well.

```plaintext
host1
    resources_available.AppC = 0
host2
    resources_available.AppC = 0
host3
    resources_available.AppC = 4
host4
    resources_available.AppC = 4
host5
    resources_available.AppC = 0
host6
    resources_available.AppC = 0
```

5.14.8 Using GPUs

You can configure PBS to support GPU scheduling. We describe how to configure both basic and advanced GPU scheduling. Basic GPU scheduling will meet the needs of most job submitters; it allows a job to request the number of GPUs it needs, as long as the job requests exclusive use of each node containing the GPUs. Advanced GPU scheduling allows jobs to request specific GPUs.

PBS Professional allocates GPUs, but does not bind jobs to any particular GPU; the application itself, or the CUDA library, is responsible for the actual binding.

5.14.8.1 Basic GPU Scheduling

Basic scheduling consists of prioritizing jobs based on site policies, controlling access to nodes with GPUs, ensuring that GPUs are not over-subscribed, and tracking use of GPUs in accounting logs.

Configuring PBS to perform basic scheduling of GPUs is relatively simple, and only requires defining and configuring a single custom resource to represent the number of GPUs on each node.

This method allows jobs to request unspecified GPUs. Jobs should request exclusive use of the node to prevent other jobs being scheduled on their GPUs.
5.14.8.2 Advanced GPU Scheduling

Advanced scheduling allows a job to separately allocate (request and/or identify) each individual GPU on a node.

In this case, both PBS and the applications themselves must support individually allocating the GPUs on a node. Advanced scheduling requires defining a PBS vnode for each GPU.

This capability is useful for sharing a single multi-GPU node among multiple jobs, where each job requires exclusive use of its GPUs.

5.14.8.3 Configuring PBS for Basic GPU Scheduling

You configure a single custom consumable resource to represent all GPU devices on an execution host. Create a host-level global consumable custom resource to represent GPUs. We recommend that the custom GPU resource is named *ngpus*. Set the value for this resource at each vnode to the number of GPUs on the vnode.

The *ngpus* resource is used exactly the way you use the *ncpus* resource.

5.14.8.3.i Example of Configuring PBS for Basic GPU Scheduling

In this example, there are two execution hosts, HostA and HostB, and each execution host has 4 GPU devices.

1. Stop the server and scheduler. On the server's host, type:
   ```
   /etc/init.d/pbs stop
   ```
2. Edit `PBS_HOME/server_priv/resourcedef`, and add the following line:
   ```
   ngpus type=long flag=nh
   ```
3. Edit `PBS_HOME/sched_priv/sched_config` to add ngpus to the list of scheduling resources:
   ```
   resources: "ncpus, mem, arch, host, vnode, ngpus"
   ```
4. Restart the server and scheduler. On the server's host, type:
   ```
   /etc/init.d/pbs start
   ```
5. Add the number of GPU devices available to each execution host in the cluster via qmgr:
   ```
   Qmgr: set node HostA resources_available.ngpus=4
   Qmgr: set node HostB resources_available.ngpus=4
   ```
5.14.8.4 Configuring PBS for Advanced GPU Scheduling

You configure each GPU device in its own vnode, and each GPU vnode has a resource to contain the device number of its GPU.

Create and set two custom resources:

• Create a host-level global consumable resource to represent the GPUs on a vnode. We recommend that this resource is called *ngpus*.
  
  Set *ngpus* on each node to the number of GPUs on that node.

• Create a host-level global non-consumable resource containing the GPU device number, which serves to tie the individual GPU to the vnode. We recommend that this resource is called *gpu_id*.
  
  Set *gpu_id* for each GPU to the device number of that GPU.
5.14.8.4.i  Example of Configuring PBS for Advanced GPU Scheduling

In this example, there is one execution host, HostA, that has two vnodes, HostA[0] and HostA[1], as well as the natural vnode. HostA has 4 CPUs, 2 GPUs, and 16 GB of memory.

1. Stop the server and scheduler. On the server's host, type:
   `/etc/init.d/pbs stop`

2. Edit `PBS_HOME/server_priv/resourcedef` to add the new custom resources:
   ```
   ngpus type=long flag=nh
   gpu_id type=string flag=h
   ```

3. Edit `PBS_HOME/sched_priv/sched_config` to add `ngpus` and `gpu_id` to the list of scheduling resources:
   ```
   resources: “ncpus, mem, arch, host, vnode, ngpus, gpu_id”
   ```

4. Restart the server and scheduler. On the server's host, type:
   `/etc/init.d/pbs start`

5. Create a vnode configuration file for each execution host where GPUs are present. The script for HostA is named `hostA_vnodes`, and is shown here:
   ```
   $configversion 2
   hostA: resources_available.ncpus = 0
   hostA: resources_available.mem = 0
   hostA[0]: resources_available.ncpus = 2
   hostA[0] : resources_available.mem = 8gb
   hostA[0] : resources_available.ngpus = 1
   hostA[0] : resources_available.gpu_id = gpu0
   hostA[0] : sharing = default_excl
   hostA[1] : resources_available.ncpus = 2
   hostA[1] : resources_available.mem = 8gb
   hostA[1] : resources_available.ngpus = 1
   hostA[1] : resources_available.gpu_id = gpu1
   hostA[1]: sharing = default_excl
   ```

6. Add vnode configuration information in the following manner, for each node with GPUs:
   ```
   PBS_EXEC/sbin/pbs_mom -s insert HostA_vnodes HostA_vnodes
   ```

7. Signal each MoM to re-read its configuration files:
   ```
   kill -HUP <pbs_mom PID>
   ```
5.14.9 Using FPGAs

You can configure a custom resource that allows PBS to track the usage of FPGAs. The FPGAs are detected outside of PBS at boot time. There are two basic methods for automatic configuration of the FPGA resource:

• Create a global static host-level resource called nfpgas. Create a boot-up script in init.d that detects the presence of the FPGAs, and sets the value of the nfpgas resource.

• Create a global dynamic host-level resource called nfpgas. This resource calls a script to detect the presence of FPGAs

We recommend the static resource, because FPGAs are static, and there is a performance penalty for a dynamic resource.

5.14.10 Custom Resource Caveats

• Because some custom resources are external to PBS, they are not completely under the control of PBS. Therefore it is possible for PBS to query and find a resource available, schedule a job to run and use that resource, only to have an outside entity take that resource before the job is able to use it. For example, say you had an external resource of “scratch space” and your local query script simply checked to see how much disk space was free. It would be possible for a job to be started on a host with the requested space, but for another application to use the free space before the job did.

• If a resource is not put in the scheduler’s resources: line, when jobs request the resource, that request will be ignored. If the resource is ignored, it cannot be used to accept or reject jobs at submission time. For example, if you create a string resource String1 on the server, and set it to foo, a job requesting “-l String1=bar” will be accepted. The only exception is host-level Boolean resources, which are considered when scheduling, whether or not they are in the scheduler’s resources: line.

• Do not create resources with the same names or prefixes that PBS uses when creating custom resources for specific systems. See “Custom Cray Resources” on page 308 of the PBS Professional Reference Guide.

• Using dynamic host-level resources can slow the scheduler down, because the scheduler must wait for each resource-query script to run.

5.14.11 Deleting Custom Resources

Removing any custom resource definition should be done with care. It is important to delete a custom resource completely and in the correct order. These steps are described below.
5.14.11.1 Alter Any Jobs Requesting Custom Resource

If you delete a resource definition from PBS_HOME/server_priv/resourcedef and restart the server, all jobs requesting that resource will be purged from the server when it is restarted. To avoid losing jobs requesting a deleted custom resource, use the qalter command on those jobs before restarting the server.

5.14.11.2 Remove References to Custom Resource

Before you delete a custom resource, you must remove all references to that resource, including where it is used in hooks, the scheduling formula, queue and server settings such as resources_available, etc. Any attributes containing the custom resource must be unset for that resource.

5.14.11.3 Steps in Deleting Custom Resource

To remove custom resources:

1. Remove all references to the resource
2. Make sure that the pbs_server daemon is running
3. Set scheduling to False
4. For each custom resource to be removed, use qmgr to unset that resource at the server, queue, or node level:
   - qmgr: unset server <resource name>
   - qmgr: unset queue <resource name>
   - qmgr: unset node <resource name>
5. Quit qmgr
6. Edit the PBS_HOME/server_priv/resourcedef file to remove the unwanted resources
7. Edit the resources: line in PBS_HOME/sched_priv/sched_config to remove the unwanted resource name
   - If the resource is a server dynamic resource, remove the resource name from the server_dyn_res: line
   - If the resource is a MoM dynamic resource, remove the resource from the
Chapter 5  PBS Resources

mom_resources:  line

8. For each MoM whose Version 2 configuration file contains references to the resource, use the `pbs_mom -s insert` command to update the Version 2 configuration file. See section 3.5.3, “Creating Version 2 MoM Configuration Files”, on page 53.

9. If the resource is a local dynamic resource, defined in the MoM Version 1 configuration file:

   For each host where the unwanted resource is defined, edit PBS_HOME/mom_priv/config and remove the resource entry line.

10. Restart the `pbs_server` daemon; see section 5.14.3.1, “Restarting the Server”, on page 332.

11. HUP each MoM; see section 5.14.3.2, “Restarting or Reinitializing MoM”, on page 332.

12. Set scheduling to `True`.

5.15 Managing Resource Usage

You can manage resource usage from different directions:

- You can manage resource usage by users, groups, and projects, and the number of jobs, at the server and queue level. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367.
  - You can manage the total amount of each resource that is used by projects, users or groups, at the server or queue level. For example, you can manage how much memory is being used by jobs in queue QueueA.
  - You can manage the number of jobs being run by projects, users or groups, at the server or queue level. For example, you can limit the number of jobs enqueued in queue QueueA by any one group to 30, and by any single user to 5.

- You can specify how much of each resource any job is allowed to use, at the server and queue level. See section 5.15.3, “Placing Resource Limits on Jobs”, on page 391 and section 5.13, “Using Resources to Restrict Server, Queue Access”, on page 318.

- You can set default limits for usage for each resource, at the server or queue level, so that jobs that do not request a given resource inherit that default, and are limited to the inherited amount. For example, you can specify that any job entering queue QueueA not specifying `walltime` is limited to using 4MB of memory. See section 5.9.3, “Specifying Job Default Resources”, on page 305.

- You can set limits on the number of jobs that can be run at each vnode by users, by groups, or overall. See section 5.15.2, “Limiting Number of Jobs at Vnode”, on page 368.
5.15.1 Managing Resource Usage By Users, Groups, and Projects, at Server & Queues

You can set separate limits for resource usage by individual users, individual groups, individual projects, generic users, generic groups, generic projects, and the total used overall. You can limit the amount of resources used, and the number of running and queued jobs. These limits can be defined separately for each queue and for the server. You define the limits by setting server and queue limit attributes. For information about projects, see section 11.3, “Grouping Jobs By Project”, on page 823.

There are two incompatible sets of server and queue limit attributes used in limiting resource usage. The first set existed in PBS Professional before Version 10.1, and we call them the old limit attributes. The old limit attributes are discussed in section 5.15.1.15, “Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1”, on page 388. The set introduced in Version 10.1 is called simply the limit attributes, and they are discussed here.

You can use either the limit attributes or the old limit attributes for the server and queues, but not both. See section 5.15.1.13.vi, “Do Not Mix Old And New Limits”, on page 387.

There is a set of limit attributes for vnodes which existed before Version 10.1 and can be used with either the limit attributes or the old limit attributes. These are discussed in section 5.15.2, “Limiting Number of Jobs at Vnode”, on page 390.

The server and queues each have per-job limit attributes which operate independently of the limits discussed in this section. The resources_min.<resource> and resources_max.<resource> server and queue attributes are limits on what each individual job may use. See section 5.13, “Using Resources to Restrict Server, Queue Access”, on page 318 and section 5.15.3, “Placing Resource Limits on Jobs”, on page 391.

5.15.1.1 Examples of Managing Resource Usage at Server and Queues

You can limit resource usage and job count for specific projects, users and groups:

- UserA can use no more than 6 CPUs, and UserB can use no more than 4 CPUs, at one time anywhere in the PBS complex.
- The crashtest group can use no more than 16 CPUs at one time anywhere in the PBS complex.
- UserC accidentally submitted 200,000 jobs last week. UserC can now have no more than
Chapter 5  PBS Resources

25 jobs enqueued at one time.

- All jobs request the server-level custom resource `nodehours`, which is used for allocation. UserA cannot use more than 40 `nodehours` in the PBS complex. Once UserA reaches the `nodehours` limit, then all queued jobs owned by UserA are not eligible for execution.

- You wish to allow UserD to use 12 CPUs but limit all other users to 4 CPUs.

- Jobs belonging to Project A can use no more than 8 CPUs at Queue1.

You can limit the number of jobs a particular project, user or group runs in a particular queue:

- UserE can use no more than 2 CPUs at one time at Queue1, and 6 CPUs at one time at Queue2.

- You wish to limit UserF to 10 running jobs in queue Queue3, but allow all other users unlimited jobs running in the same queue.

- UserG is a member of Group1. You have a complex-wide limit of 5 running jobs for UserG. You have a limit at Queue1 of 10 running jobs for Group1. This way, up to 10 of the running jobs in Queue1 can belong to Group1, and 5 of these can belong to UserG.

- UserH is a member of Group1. You have a complex-wide limit of 5 running jobs for UserH. You have a limit at Queue1 of 10 running jobs for any group in Queue1. This way, no group in Queue1 can run more than 10 jobs total at one time, and 5 of these can belong to UserH.

- UserJ is a member of Group1. You have a complex-wide limit of 10 running jobs for UserJ. You also have a limit at Queue1 of 5 running jobs for Group1. This means that there may be up to 5 running jobs owned by users belonging to Group1 in Queue1, and up to 5 of these can be owned by UserJ. UserJ can also have another 5 running jobs owned by Group1 in any other queue, or owned by a different group in Queue1.

- No more than 12 jobs belonging to Project A can run at Queue1, and all other projects are limited to 8 jobs at Queue1.

You can ensure fairness in the use of resources:

- You have multiple departments which have shared the purchase of a large Altix. Each department would like to ensure fairness in the use of the Altix, by setting limits on individual users and groups.

- You have multiple departments, each of which purchases its own machines. Each department would like to limit the use of its machines so that all departmental users have specific limits. In addition, each department would like to allow non-departmental users to use its machines when they are under-utilized, while giving its own users priority on its machines. A non-departmental user can run jobs on a departmental machine, as long as no departmental users’ jobs are waiting to run.
5.15.1.2 Terminology

Limit
The maximum amount of a resource that can be consumed at any time by running jobs or allocated to queued jobs, or the maximum number of jobs that can be running, or the maximum number of jobs that can be queued.

Overall limit
Limit on the total usage. In the context of server limits, this is the limit for usage at the PBS complex. In the context of queue limits, this is the limit for usage at the queue. An overall limit is applied to the total usage at the specified location. Separate overall limits can be specified at the server and each queue.

Generic user limit
Applies separately to users at the server or a queue. The limit for users who have no individual limit specified. A separate limit for generic users can be specified at the server and at each queue.

Generic group limit
Applies separately to groups at the server or a queue. The limit for groups which have no individual limit specified. A limit for generic groups is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Generic project limit
Applies separately to projects at the server or a queue. The limit for projects which have no individual limit specified. A limit for generic projects is applied to the usage across the entire project. A separate limit can be specified at the server and each queue.

Individual user limit
Applies separately to users at the server or a queue. Limit for users who have their own individual limit specified. A limit for an individual user overrides the generic user limit, but only in the same context, for example, at a particular queue. A separate limit can be specified at the server and each queue.

Individual group limit
Applies separately to groups at the server or a queue. Limit for a group which has its own individual limit specified. An individual group limit overrides the generic group limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.
Chapter 5  

PBS Resources

**Individual project limit**
Applies separately to projects at the server or a queue. Limit for a project which has its own individual limit specified. An individual project limit overrides the generic project limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire project. A separate limit can be specified at the server and each queue.

**User limit**
A limit placed on one or more users, whether generic or individual.

**Group limit**
This is a limit applied to the total used by a group, whether the limit is a generic group limit or an individual group limit.

**Project**
In PBS, a project is a way to group jobs independently of users and groups. A project is a tag that identifies a set of jobs. Each job’s `project` attribute specifies the job’s project.

**Project limit**
This is a limit applied to the total used by a project, whether the limit is a generic project limit or an individual project limit.

**Queued jobs**
In a queue, queued jobs are the jobs that are waiting in that queue and the jobs that are running in that queue.

### 5.15.1.3 Difference Between PBS_ALL and PBS_GENERIC

Note the very important *difference* between the *overall limit* and a *generic* limit. We will describe how this works for uses, but this applies to other entities as well. You set PBS_ALL for an overall limit on the total usage of that resource by all entities, whereas you set PBS GENERIC for a limit for any single generic user.

**Example 5-12: Difference between overall limit and generic user limit**

Given the following:
- The overall server limit for running jobs is 100
- The server limit for generic users is 10
- The individual limit for User1 is 12 jobs

This means:
- Generic users (any single user except User1) can run no more than 10 jobs at this
5.15.1.4 **Hard and Soft Limits**

Hard limits are limits which cannot be exceeded. Soft limits are limits which mark the point where a project, user or group is using “extra, but acceptable” amounts of a resource. When this happens, the jobs belonging to that project, user or group are eligible for preemption. See section 4.8.33, “Using Preemption”, on page 228. Soft limits are discussed in section 4.8.33.5.i, “The Soft Limits Preemption Level”, on page 231.

5.15.1.5 **Scope of Limits at Server and Queues**

Each of the limits described above can be set separately at the server and at each queue. Each limit’s scope is the PBS object where it is set. The individual and generic project, user and group limits that are set within one scope interact with each other only within that scope. For example, a limit set at one queue has no effect at another queue.

The scope of limits set at the server encompasses queues, so that the minimum, more restrictive limit of the two is applied. For precedence within a server or queue, see section 5.15.1.7, “Precedence of Limits at Server and Queues”, on page 374.

5.15.1.6 **Ways To Limit Resource Usage at Server and Queues**

You can create a complete set of limits at the server, and you can create another complete set of limits at each queue. You can set hard and soft limits. See section 4.8.33.5.i, “The Soft Limits Preemption Level”, on page 231. You can limit resource usage at the server and the queue level for the following:

- Running jobs
  - Number of running jobs
  - Number of running jobs (soft limit)
  - Amount of each resource allocated for running jobs
  - Amount of each resource allocated for running jobs (soft limit)
- Queued jobs (this means both jobs that are waiting to run and jobs that are running from that queue)
  - Number of queued jobs (includes jobs running from that queue)
  - Amount of each resource allocated for queued jobs (includes amount of each resource allocated to jobs running from that queue)
These limits can be applied to the following:

- The total usage at the server
- The total usage at each queue
- Amount used by a single user
  - Generic users
  - Individual users
- Amount used by a single group
  - Generic groups
  - Individual groups
- Amount used by a single project
  - Generic projects
  - Individual projects

### 5.15.1.6.i Limits at Queues

Routing queues do not run jobs, so you cannot set a limit for the number of running jobs, or the amount of resources being used by running jobs, at a routing queue.

When queued jobs are counted for a queue, the number includes the jobs that are running from that queue. Jobs queued at a queue are counted the same whether they were submitted to that queue via the `qsub` command or its equivalent API, moved to that queue via the `qmove` command or its equivalent API, or routed to that queue from another queue.

When resources allocated to queued jobs are summed at a queue, the number includes the resources allocated to jobs running from that queue.

### 5.15.1.6.ii Generic and Individual Limits

You can set a generic limit for groups, so that each group must obey the same limit. You can likewise set a generic limit for users and projects. Each generic limit can be set separately at the server and at each queue. For example, if you have two queues, the generic limit for the number of jobs a user can run be 4 at QueueA and 6 at QueueB.

You can set a different individual limit for each user, and you can set individual limits for groups and for projects. Each user, group, and project can have a different individual limit at the server and at each queue.
You can use a combination of generic and individual project, user or group limits, at the server and at each queue. Within the scope of the server or a queue, all projects, users or groups except the ones with the individual limits must obey the generic limit, and the individual limits override the generic limits.

Example 5-13: Generic and individual user limits on running jobs at QueueA and QueueB

At QueueA:
- At QueueA, the generic user limit is 5
- At QueueA, Bob’s individual limit is 8
- Tom has no individual limit set at QueueA; the generic limit applies

At QueueB:
- At QueueB, the generic user limit is 2
- At QueueB, Tom’s individual limit is 1
- Bob has no individual limit at QueueB; the generic limit applies

This means:
- Bob can run 8 jobs at QueueA
- Bob can run 2 jobs at QueueB
- Tom can run 5 jobs at QueueA
- Tom can run 1 job at QueueB

5.15.1.6.iii Overall Limits

The overall limit places a cap on the total amount of the resource that can be used within the scope in question (server or queue), regardless of whether project, user, or group limits have been reached. A project, user, or group at the server or a queue cannot use any more of a resource for which the overall limit has been reached, even if that project, user, or group limit has not been reached.

Example 5-14: Overall limit at server

Given the following:
- Overall server limit on running jobs is 100
- Bob’s user limit is 10 running jobs
- 98 jobs are already running
- Bob is running zero jobs

This means:
- Bob can start only 2 jobs
### 5.15.1.7 Precedence of Limits at Server and Queues

#### 5.15.1.7.i Interactions Between Limits Within One Scope

Within the scope of a PBS object (server or queue), there is an order of precedence for limits when more than one applies to a job. The order of precedence for the limits at a queue is the same as the order at the server. The following table shows how limits interact within one scope:

#### Table 5-8: Limit Interaction Within One Scope

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual User</td>
<td>Individual user</td>
<td>Individual user</td>
<td>More restrictive</td>
<td>More restrictive</td>
<td>More restrictive</td>
<td>More restrictive</td>
</tr>
<tr>
<td>Generic User</td>
<td>Individual user</td>
<td>Generic user</td>
<td>More restrictive</td>
<td>More restrictive</td>
<td>More restrictive</td>
<td>More restrictive</td>
</tr>
<tr>
<td>Individual Group</td>
<td>More restrictive</td>
<td>More restrictive</td>
<td>Individual group</td>
<td>Individual group</td>
<td>More restrictive</td>
<td>More restrictive</td>
</tr>
<tr>
<td>Generic Group</td>
<td>More restrictive</td>
<td>More restrictive</td>
<td>Individual group</td>
<td>Generic group</td>
<td>More restrictive</td>
<td>More restrictive</td>
</tr>
<tr>
<td>Individual Project</td>
<td>More restrictive</td>
<td>More restrictive</td>
<td>More restrictive</td>
<td>Individual project</td>
<td>Individual project</td>
<td></td>
</tr>
<tr>
<td>Generic Project</td>
<td>More restrictive</td>
<td>More restrictive</td>
<td>More restrictive</td>
<td>Individual project</td>
<td>Generic project</td>
<td></td>
</tr>
</tbody>
</table>

An individual user limit overrides a generic user limit.

**Example 5-15:** Individual user limit overrides generic user limit

Given the following:
- Bob has a limit of 10 running jobs
- The generic limit is 5

This means:
- Bob can run 10 jobs
An individual group limit overrides a generic group limit in the same manner as for users. If the limits for a user and the user’s group are different, the more restrictive limit applies.

Example 5-16: More restrictive user or group limit applies

Given the following:
- Tom’s user limit for running jobs is 8
- Tom’s group limit is 7

This means:
- Tom can run only 7 jobs in that group

If a user belongs to more than one group, that user can run jobs up to the lesser of his user limit or the sum of the group limits.

Example 5-17: User can run jobs in more than one group

Given the following:
- Tom’s user limit is 10 running jobs
- GroupA has a limit of 2 and GroupB has a limit of 4
- Tom belongs to GroupA and GroupB

This means:
- Tom can run 6 jobs, 2 in GroupA and 4 in GroupB

An individual project limit overrides a generic project limit, similar to the way user and group limits work.

Project limits are applied independently of user and group limits.

Example 5-18: Project limits are applied without regard to user and group limits

Given the following:
- Project A has a limit of 2 jobs
- Bob has an individual limit of 4 jobs
- Bob’s group has a limit of 6 jobs
- Bob is running 2 jobs, both in Project A

This means:
- Bob cannot run any more jobs in Project A
5.15.1.7.ii  Interactions Between Queue and Server Limits

If the limits for a queue and the server are different, the more restrictive limit applies.

Example 5-19: More restrictive queue or server limit applies

Given the following:
- Server limit on running jobs for generic users is 10
- Queue limit for running jobs from QueueA for generic users is 15
- Queue limit for running jobs from QueueB for generic users is 5

This means:
- Generic users at QueueA can run 10 jobs
- Generic users at QueueB can run 5 jobs

Example 5-20: More restrictive queue or server limit applies

Given the following:
- Bob’s user limit on running jobs, set on the server, is 7
- Bob’s user limit on running jobs, set on QueueA, is 6

This means:
- Bob can run 6 jobs from QueueA

5.15.1.8  Resource Usage Limit Attributes for Server and Queues

Each of the following attributes can be set at both the server and each queue:

- `max_run`
  The maximum number of jobs that can be running.

- `max_run_soft`
  The soft limit on the maximum number of jobs that can be running.

- `max_run_res.<resource>`
  The maximum amount of the specified resource that can be allocated to running jobs.

- `max_run_res_soft.<resource>`
  The soft limit on the amount of the specified resource that can be allocated to running jobs.

- `max_queued`
  The maximum number of jobs that can be queued and running. Any job running from a queue is counted as being queued in that queue. At the server level, this
includes all jobs in the complex. Queueing a job includes the `qsub` and `qmove` commands and the equivalent APIs.

**max_queued_res.<resource>**

The maximum amount of the specified resource that can be allocated to queued and running jobs. Any job running from a queue is counted as being queued in that queue. At the server level, this includes all jobs in the complex. Queueing a job includes the `qsub` and `qmove` commands and the equivalent APIs.

Each attribute above can be used to specify all of the following:

- An overall limit (at the queue or server)
- A limit for generic users
- Individual limits for specific users
- A limit for generic projects
- Individual limits for specific projects
- A limit for generic groups
- Individual limits for specific groups
For example, you can specify the limits for the number of running jobs:

- **In the complex:**
  - The overall server limit (all usage in the entire complex) is **10,000**
  - The limit for generic users is **5**
  - The limit for Bob is **10**
  - The limit for generic groups is **50**
  - The limit for group GroupA is **75**
  - The limit for generic projects is **25**
  - The limit for Project A is **35**

- **At QueueA:**
  - The overall queue limit (all usage in QueueA) is **200**
  - The limit for generic users is **2**
  - The limit for Bob is **1**
  - The limit for generic groups is **3**
  - The limit for group GroupA is **7**
  - The limit for generic projects is **10**
  - The limit for Project A is **15**

- **At QueueB:**
  - The overall queue limit (all usage in QueueB) is **500**
  - The limit for generic users is **6**
  - The limit for Bob is **8**
  - The limit for generic groups is **15**
  - The limit for group GroupA is **11**
  - The limit for generic projects is **20**
  - The limit for Project A is **30**

### 5.15.1.9 How to Set Limits at Server and Queues

You can set, add, and remove limits by using the `qmgr` command to set limit attributes.
5.15.1.9.i Syntax

Format for setting a limit attribute:

\[
\text{set server } \langle \text{limit attribute}\rangle = "[\text{limit-spec}=<\text{limit}>], [\text{limit-spec}=<\text{limit}>],..."
\]

\[
\text{set } \langle \text{queue} \rangle \langle \text{queue name} \rangle \langle \text{limit attribute}\rangle = "[\text{limit-spec}=<\text{limit}>], [\text{limit-spec}=<\text{limit}>],..."
\]

Format for adding a limit to an attribute:

\[
\text{set server } \langle \text{limit attribute}\rangle += "[\text{limit-spec}=<\text{limit}>], [\text{limit-spec}=<\text{limit}>],..."
\]

\[
\text{set } \langle \text{queue} \rangle \langle \text{queue name} \rangle \langle \text{limit attribute}\rangle += "[\text{limit-spec}=<\text{limit}>], [\text{limit-spec}=<\text{limit}>],..."
\]

Format for removing a limit from an attribute; note that the value for \langle limit\rangle need not be specified when removing a limit:

\[
\text{set server } \langle \text{limit attribute}\rangle -= "[\text{limit-spec}], [\text{limit-spec}],..."
\]

\[
\text{set } \langle \text{queue} \rangle \langle \text{queue name} \rangle \langle \text{limit attribute}\rangle -= "[\text{limit-spec}], [\text{limit-spec}],..."
\]

Alternate format for removing a limit from an attribute; note that the value of \langle limit\rangle used when removing a limit must match the value of the limit:

\[
\text{set server } \langle \text{limit attribute}\rangle -= "[\text{limit-spec}=<\text{limit}>], [\text{limit-spec}=<\text{limit}>],..."
\]

\[
\text{set } \langle \text{queue} \rangle \langle \text{queue name} \rangle \langle \text{limit attribute}\rangle -= "[\text{limit-spec}=<\text{limit}>], [\text{limit-spec}=<\text{limit}>],..."
\]

where limit-spec specifies a user limit, a group limit, or an overall limit:

<table>
<thead>
<tr>
<th>Limit</th>
<th>limit-spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall limit</td>
<td>o:PBS_ALL</td>
</tr>
<tr>
<td>Generic users</td>
<td>u:PBS_GENERIC</td>
</tr>
<tr>
<td>An individual user</td>
<td>u:&lt;username&gt;</td>
</tr>
<tr>
<td>Generic groups</td>
<td>g:PBS_GENERIC</td>
</tr>
<tr>
<td>An individual group</td>
<td>g:&lt;group name&gt;</td>
</tr>
<tr>
<td>Generic projects</td>
<td>p:PBS_GENERIC</td>
</tr>
<tr>
<td>An individual project</td>
<td>p:&lt;project name&gt;</td>
</tr>
</tbody>
</table>
Chapter 5  PBS Resources

The limit-spec can contain spaces anywhere except after the colon (";").

If there are comma-separated limit-specs, the entire string must be enclosed in double quotes.

A username, group name, or project name containing spaces must be enclosed in quotes.

If a username, group name, or project name is quoted using double quotes, and the entire string requires quotes, the outer enclosing quotes must be single quotes. Similarly, if the inner quotes are single quotes, the outer quotes must be double quotes.

PBS_ALL is a keyword which indicates that this limit applies to the usage total.

PBS GENERIC is a keyword which indicates that this limit applies to generic users or groups.

When removing a limit, the limit value does not need to be specified.

PBS_ALL and PBS_GENERIC are case-sensitive.

5.15.1.9.ii  Examples of Setting Server and Queue Limits

Example 5-21: To set the maxqueued limit on QueueA to 5 for total usage, and to limit user bill to 3:

```
Qmgr: s q QueueA max_queued = "[o:PBS_ALL=5], [u:bill =3]"
```

Example 5-22: On QueueA, set the maximum number of CPUs and the maximum amount of memory that user bill can request in his queued jobs:

```
Qmgr: s q QueueA max_queued_res.ncpus ="[u:bill=5]",
      max_queued_res.mem = "[u:bill=100mb]"
```

Example 5-23: To set a limit for a username with a space in it, and to set a limit for generic groups:

```
Qmgr: s q QueueA max_queued = '[u:"\PROG\Named User" = 1],
      [g:PBS_GENERIC=4]'
```

Example 5-24: To set a generic server limit for projects, and an individual server limit for Project A:

```
Qmgr: set server max_queued = '[p:PBS_GENERIC=6], [p:ProjectA=8]'
```
5.15.1.9.iii  Examples of Adding Server and Queue Limits

Example 5-25: To add an overall limit for the maximum number of jobs that can be queued at QueueA to 10:

```
Omgr: s q QueueA max_queued += [o:PBS_ALL=10]
```

Example 5-26: To add an individual user limit, an individual group limit, and a generic group limit on queued jobs at QueueA:

```
Omgr: s q QueueA max_queued += "[u:user1= 5],
[g:GroupMath=5],[g:PBS GENERIC=2]"
```

Example 5-27: To add a limit at QueueA on the number of CPUs allocated to queued jobs for an individual user, and a limit at QueueA on the amount of memory allocated to queued jobs for an individual user:

```
Omgr: s q QueueA max_queued_res.ncpus += [u:tom=5],
max_queued_res.mem += [u:tom=100mb]
```

Example 5-28: To add an individual server limit for Project B:

```
Omgr: set server max_queued += [p:ProjectB=4]
```

5.15.1.9.iv  Examples of Removing Server and Queue Limits

It is not necessary to specify the value of the limit when removing a limit, but you can specify the value of the limit.

Example 5-29: To remove the generic user limit at QueueA for queued jobs, use either of the following:

```
Omgr: set queue QueueA max_queued -= [u:PBS GENERIC]
Omgr: set queue QueueA max_queued -= [u:PBS GENERIC=2]
```

Example 5-30: To remove the limit on queued jobs at QueueA for Named User, use either of the following:

```
Omgr: set queue QueueA max_queued -= [u:"\PROG\Named User"]
Omgr: set queue QueueA max_queued -= [u:"\PROG\Named User"=1]
```

Example 5-31: To remove the limit at QueueA on the amount of memory allocated to an individual user, use either of the following:

```
Omgr: set queue QueueA max_queued_res.mem -= [u:tom]
Omgr: set queue QueueA max_queued_res.mem -= [u:tom=100mb]
```
To remove the limit on the number of CPUs allocated to queued jobs for user bill, use either of the following:

```bash
Qmgr: set queue QueueA max_queued_res.ncpus -= [u:bill]
Qmgr: set queue QueueA max_queued_res.ncpus -= [u:bill=5]
```

Example 5-32: To remove a generic user limit and an individual user limit, use either of the following:

```bash
Qmgr: set queue QueueA max_queued -= "[u:user1], [u:PBS_GENERIC]"
Qmgr: set queue QueueA max_queued -= "[u:user1=2],
[u:PBS GENERIC=4]"
```

Example 5-33: To remove the individual server limit for Project B, use either of the following:

```bash
Qmgr: set server max_queued -=[p:ProjectB]
Qmgr: set server max_queued -=[p:ProjectB=4]
```

### 5.15.1.10 Who Can Set Limits at Server and Queues

As with other server and queue attributes, only PBS Managers and Operators can set limit attributes.
5.15.1.11 Viewing Server and Queue Limit Attributes

5.15.1.11.i Printing Server and Queue Limit Attributes

You can use the `qmgr` command to print the commands used to set the limit attributes at the server or queue.

Example 5-34: To print all the limit attributes for queue QueueA:

```bash
Qmgr: p q QueueA max_queued, max_queued_res
#
# Create queues and set their attributes.
#
# Create and define queue QueueA
#
create queue QueueA
set queue QueueA max_queued = "[o:PBS_ALL=10]"
set queue QueueA max_queued += "[u:PBS_GENERIC=2]"
set queue QueueA max_queued += "[u:bill=3]"
set queue QueueA max_queued += "[u:tom=15]"
set queue QueueA max_queued += "[u:user1=3]"
set queue QueueA max_queued += '[:PROG\Named User"=1]'
set queue QueueA max_queued += "[g:PBS_GENERIC=2]"
set queue QueueA max_queued += "[g:GroupMath=5]"
set queue QueueA max_queued_res.ncpus = "[u:bill=5]"
set queue QueueA max_queued_res.ncpus += "[u:tom=5]"
set queue QueueA max_queued_res.mem = "[u:bill=100mb]"
set queue QueueA max_queued_res.mem += "[u:tom=100mb]"
```
5.15.1.11.i Listing Server and Queue Limit Attributes

You can use the `qmgr` command to list the limit attributes for the queue or server.

Example 5-35: To list the `max_queued` and `max_queued_res` attributes for QueueA:

```
Qmgr: l q QueueA max_queued, max_queued_res
Queue: QueueA
  max_queued = [o:PBS_ALL=10]
  max_queued = [g:PBS_GENERIC=2]
  max_queued = [g:GroupMath=5]
  max_queued = [u:PBS_GENERIC=2]
  max_queued = [u:bill=3]
  max_queued = [u:tom=15]
  max_queued = [u:user1=3]
  max_queued = [u:"\PROG\Named User"=1]
  max_queued_res.ncpus = [u:bill=5]
  max_queued_res.ncpus = [u:tom=5]
  max_queued_res.mem = [u:bill=5]
  max_queued_res.mem = [u:bill=100mb]
  max_queued_res.mem = [u:tom=100mb]
```
5.15.1.11.iii Using the \texttt{qstat} Command to View Queue Limit Attributes

You can use the \texttt{qstat} command to see the limit attribute settings for the queue or server.

Example 5-36: To see the settings for the \texttt{max\_queued} and \texttt{max\_queued\_res} limit attributes for QueueA using the \texttt{qstat} command:

\begin{verbatim}
qstat -Qf QueueA
Queue: QueueA
...
max\_queued = [o:PBS\_ALL=10]
max\_queued = [g:PBS\_GENERIC=2]
max\_queued = [g:GroupMath=5]
max\_queued = [u:PBS\_GENERIC=2]
max\_queued = [u:bill=3]
max\_queued = [u:tom=3]
max\_queued = [u:cs=3]
max\_queued = [u:"\PROG\Named\ User"=1]
max\_queued\_res.ncpus = [u:bill=5]
max\_queued\_res.ncpus = [u:tom=5]
max\_queued\_res.mem = [u:bill=5]
max\_queued\_res.mem = [u:bill=100mb]
max\_queued\_res.mem = [u:tom=100mb]
\end{verbatim}

5.15.1.12 How Server and Queue Limits Work

Affected jobs are jobs submitted by the user or group, or jobs belonging to a project, whose limit has been reached. The following table shows what happens when a given limit is reached:

<table>
<thead>
<tr>
<th>Limit</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running jobs</td>
<td>No more affected jobs are run at this server or queue until the number of affected running jobs drops below the limit.</td>
</tr>
<tr>
<td>Queued jobs</td>
<td>The queue does not accept any more affected jobs until the number of affected queued jobs drops below the limit. Affected jobs submitted directly to the queue are rejected. Affected jobs in a routing queue whose destination is this queue remain in the routing queue.</td>
</tr>
</tbody>
</table>
Chapter 5  

PBS Resources

Table 5-10: Actions Performed When Limits Are Reached

<table>
<thead>
<tr>
<th>Limit</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources for running jobs</td>
<td>The queue does not run any more affected jobs until the limit would not be exceeded if the next affected job were to start.</td>
</tr>
<tr>
<td>Resources for queued jobs</td>
<td>The queue does not accept any more affected jobs until the limit would not be exceeded if the next affected job were to start. Affected jobs submitted directly to the queue are rejected. Affected jobs in a routing queue whose destination is this queue remain in the routing queue.</td>
</tr>
</tbody>
</table>

5.15.1.13  Caveats and Advice for Server and Queue Limits

5.15.1.13.i  Queued Jobs

The term queued jobs means, for a queue, all the jobs that are in that queue waiting to run, as well as all the jobs currently running from that queue.

The total number of queued jobs at the server level is the total number of queued and running jobs in the complex. This includes jobs in routing queues.

5.15.1.13.ii  Avoiding Overflow

On PBS server platforms for which the native size of a long is less than 64 bits, you should refrain from defining a limit on a resource of type long whose cumulative sum over all queued jobs would exceed the storage capacity of the resource variable. For example, if each submitted job were to request 100 hours of the cput resource, overflow would occur on a 32-bit platform when 5965 jobs (which is $\left(2^{31} - 1\right)/360000$ seconds) were queued.

5.15.1.13.iii  Ensuring That Limits Are Effective

In order for limits to be effective, each job must specify each limited resource. This can be accomplished using defaults; see section 5.9.3, “Specifying Job Default Resources”, on page 305. You can also use hooks; see section “Hooks”, on page 413.

5.15.1.13.iv  Array Jobs

An array job with N subjobs is considered to consume N times the amount of resources requested when it was submitted. For example, if there is a server limit of 100 queued jobs, no user would be allowed to submit an array job with more than 100 subjobs.
5.15.1.13.v Avoiding Job Rejection

Jobs are rejected when users, groups, or projects who have reached their limit submit a job in the following circumstances:

- The job is submitted to the execution queue where the limit has been reached
- The job is submitted to the complex, and the server limit has been reached

If you wish to avoid having jobs be rejected, you can set up a routing queue as the default queue. Set the server’s `default_queue` attribute to the name of the routing queue. See section 2.2.6, “Routing Queues”, on page 24.

5.15.1.13.vi Do Not Mix Old And New Limits

The new limit attributes are incompatible with the old limit attributes. See section 5.15.1.15, “Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1”, on page 388. You cannot mix the use of old and new resource usage limit attributes. This means that:

- If any old limit attribute is set, and you try to set a new limit attribute, you will get error 15141.
- If any new limit attribute is set, and you try to set an old limit attribute, you will get error 15141.

You must unset all of one kind in order to set any of the other kind.

5.15.1.13.vii Do Not Limit Running Time

Beware creating limits such as `max_run_res.walltime` or `max_run_res.max_walltime`. The results probably will not be useful. You will be limiting the amount of walltime that can be requested by running jobs for a user, group, or project. For example, if you set a walltime limit of 10 hours for group A, then group A cannot run one job requesting 5 hours and another job requesting 6 hours.

5.15.1.14 Errors and Logging for Server and Queue Limits

5.15.1.14.i Error When Setting Limit Attributes

Attempting to set a new limit attribute while an old limit attribute is set:

- "use new/old qmgr syntax, not both"
- "Attribute name <new> not allowed. Older name <old> already set"

Attempting to set an old limit attribute while a new limit attribute is set:

- "use new/old qmgr syntax, not both"
- "Attribute name <old> not allowed: Newer name <new> already set"
5.15.1.14.ii Logging Events

Whenever a limit attribute is set or modified, the server logs the event, listing which attribute was modified and who modified it.

Whenever a limit is reached, and would be exceeded by a job, the scheduler logs the event, listing the limit attribute and the reason.

5.15.1.14.iii Queued Limit Error Messages

When a limit for queued jobs or resources allocated to queued jobs is reached, the command involved presents a message. This command can be qsub, qmove or qalter.

5.15.1.14.iv Run Limit Error Messages

See “Run Limit Error Messages” on page 441 of the PBS Professional Reference Guide for a list of run limit error messages.

5.15.1.15 Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1

The old server and queue limit attributes discussed here existed in PBS Professional before Version 10.1. The old limit attributes continue to function as they did in PBS Professional 10.0. These attributes are incompatible with the limit attributes introduced in Version 10.1. See section 5.15.1.13.vi, “Do Not Mix Old And New Limits”, on page 387 and section 5.15.1.14.i, “Error When Setting Limit Attributes”, on page 387. These limits are compatible with the limits discussed in section 5.15.2, “Limiting Number of Jobs at Vnode”, on page 390.

The following table shows how the old limit attributes are used:

<table>
<thead>
<tr>
<th>Limit</th>
<th>Overall Limit</th>
<th>Generic Users</th>
<th>Generic Groups</th>
<th>Individual Users</th>
<th>Individual Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of running jobs</td>
<td>max_run</td>
<td>max_user_run</td>
<td>max_group_run</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Maximum number of running jobs (soft limit)</td>
<td>N/A</td>
<td>max_user_run_soft</td>
<td>max_group_run_soft</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
5.15.1.15.i  Precedence of Old Limits

If an old limit is defined at both the server and queue, the more restrictive limit applies.

5.15.1.15.ii  Old Server Limits

For details of these limits, see “Server Attributes” on page 316 of the PBS Professional Reference Guide.

max_running
The maximum number of jobs allowed to be selected for execution at any given time.

max_group_res, max_group_res_soft
The maximum amount of the specified resource that all members of the same UNIX group may consume simultaneously.

max_group_run, max_group_run_soft
The maximum number of jobs owned by a UNIX group that are allowed to be running from this server at one time.
max_user_res,
max_user_res_soft
The maximum amount of the specified resource that any single user may consume.

max_user_run,
max_user_run_soft
The maximum number of jobs owned by a single user that are allowed to be running at one time.

5.15.1.15.iii  Old Queue Limits
For details of these limits, see “Queue Attributes” on page 354 of the PBS Professional Reference Guide.

max_group_res,
max_group_res_soft
The maximum amount of the specified resource that all members of the same UNIX group may consume simultaneously, in the specified queue.

max_group_run,
max_group_run_soft
The maximum number of jobs owned by a UNIX group that are allowed to be running from this queue at one time.

max_queuable
The maximum number of jobs allowed to reside in the queue at any given time. Once this limit is reached, no new jobs will be accepted into the queue.

max_user_res,
max_user_res_soft
The maximum amount of the specified resource that any single user may consume in submitting to this queue.

max_user_run,
max_user_run_soft
The maximum number of jobs owned by a single user that are allowed to be running at one time from this queue.

5.15.2  Limiting Number of Jobs at Vnode
You can set limits at each vnode on the number of jobs that can be run by any user, by any group, or by everyone taken together. You set these limits by specifying values for vnode attributes. They are listed here:
max_group_run

The maximum number of jobs owned by any users in a single group that are allowed to be run on this vnode at one time.

Format: integer

Qmgr: set node MyNode max_group_run=8

max_running

The maximum number of jobs allowed to be run on this vnode at any given time.

Format: integer

Qmgr: set node MyNode max_running=22

max_user_run

The maximum number of jobs owned by a single user that are allowed to be run on this vnode at one time.

Format: integer

Qmgr: set node MyNode max_user_run=4

5.15.3 Placing Resource Limits on Jobs

Jobs are assigned limits on the amount of resources they can use. Each limit is set at the amount requested or allocated by default. These limits apply to how much the job can use on each vnode (per-chunk limit) and to how much the whole job can use (job-wide limit). Limits are derived from both requested resources and applied default resources. For information on default resources, see section 5.9.3, “Specifying Job Default Resources”, on page 305.

Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are the amount of per-chunk resources requested, both from explicit requests and from defaults.

The consumable resources requested for chunks in the select specification are summed, and this sum makes a job-wide limit. Job resource limits from sums of all chunks override those from job-wide defaults and resource requests.

Job resource limits set a limit for per-job resource usage. Various limit checks are applied to jobs. If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated.
5.15.3.1 How Limits Are Derived

Job resource limits are derived in this order from the following:

1. Explicitly requested job-wide resources (e.g. -l resource=value)
2. The following built-in chunk-level resources in the job’s select specification (e.g. -l select =...)
   - accelerator_memory
   - mem
   - mpiprocs
   - naccelerators
   - ncpus
   - netwins
   - nodect
   - vmem
3. The server’s default_qsub_arguments attribute
4. The queue’s resources_default.<resource>
5. The server’s resources_default.<resource>
6. The queue’s resources_max.<resource>
7. The server’s resources_max.<resource>

The server’s default_chunk.<resource> does not affect job-wide limits.

You can use a hook to set a per-chunk limit, using any hook that operates on jobs, such as a job submission hook, a modify job hook, etc.

5.15.3.2 Configuring Per-job Limits at Server and Queue

You can set per-job limits on the amount of each resource that any one job can use. You can set these limits at the server and at each queue. For example, you can specify the following limits:

- Jobs at the server can use no more than 48 hours of CPU time
- Jobs at QueueA can use no more than 12 hours of CPU time
- Jobs at QueueA must request more than 2 hours of CPU time
To set these limits, specify values for the server’s `resources_max.<resource>` attribute and each queue’s `resources_max.<resource>` and `resources_min.<resource>` attributes. The server does not have a `resources_min.<resource>` attribute. To set the maximum at the server, the format is:

```
Qmgr: set server resources_max.<resource> = value
```

To set the maximum and minimum at the queue, the format is:

```
Qmgr: set queue <queue name> resources_max.<resource> = value
Qmgr: set queue <queue name> resources_min.<resource> = value
```

For example, to set the 48 hour CPU time limit:

```
Qmgr: set server resources_max.cput = 48:00:00
```

### 5.15.3.2.i Running Time Limits at Server and Queues

For non-shrink-to-fit jobs, you can set limits on `walltime` at the server or queue. To set a `walltime` limit for non-shrink-to-fit jobs at the server or a queue, use `resources_max.walltime` and `resources_min.walltime`.

For shrink-to-fit jobs, running time limits are applied to `max_walltime` and `min_walltime`, not `walltime`. To set a running time limit for shrink-to-fit jobs, you cannot use `resources_max` or `resources_min` for `max_walltime` or `min_walltime`. Instead, use `resources_max.walltime` and `resources_min.walltime`. See section 4.8.41.6, “Shrink-to-fit Jobs and Resource Limits”, on page 267.

### 5.15.3.3 Configuring Per-job Resource Limit Enforcement at Vnodes

For a job, enforcement of resource limits is per-MoM, not per-vnode. So if a job requests 3 chunks, each of which has 1MB of memory, and all chunks are placed on one host, the limit for that job for memory for that MoM is 3MB. Therefore one chunk can be using 2 MB and the other two using 0.5MB and the job can continue to run.

Job resource limits can be enforced for single-vnode jobs, or for multi-vnode jobs that are using LAM or a PBS-aware MPI. See the following table for an overview. Memory limits are handled differently depending on the operating system. See "Job Memory Limit Enforce-
ment on UNIX” on page 395. The ncpus limit can be adjusted in several ways. See “Job ncpus Limit Enforcement” on page 397 for a discussion. The following table summarizes how resource limits are enforced at vnodes:

Table 5-12: Resource Limit Enforcement at Vnodes

<table>
<thead>
<tr>
<th>Limit</th>
<th>What determines when limit is enforced</th>
<th>Scope of limit</th>
<th>Enforcement method</th>
</tr>
</thead>
<tbody>
<tr>
<td>file size</td>
<td>automatically</td>
<td>per-process</td>
<td>setrlimit()</td>
</tr>
<tr>
<td>vmem</td>
<td>If job requests or inherits vmem</td>
<td>job-wide</td>
<td>MoM poll</td>
</tr>
<tr>
<td>pvmem</td>
<td>If job requests or inherits pvmem</td>
<td>per-process</td>
<td>setrlimit()</td>
</tr>
<tr>
<td>pmem</td>
<td>If job requests or inherits pmem</td>
<td>per-process</td>
<td>setrlimit()</td>
</tr>
<tr>
<td>pcpus</td>
<td>If job requests or inherits pcpus</td>
<td>per-process</td>
<td>setrlimit()</td>
</tr>
<tr>
<td>cput</td>
<td>If job requests or inherits cput</td>
<td>job-wide</td>
<td>MoM poll</td>
</tr>
<tr>
<td>walltime</td>
<td>If job requests or inherits walltime</td>
<td>job-wide</td>
<td>MoM poll</td>
</tr>
<tr>
<td>mem</td>
<td>if $enforce mem in MoM’s config</td>
<td>job-wide</td>
<td>MoM poll</td>
</tr>
<tr>
<td>ncpus</td>
<td>if $enforce cpuaverage, $enforce cpuburst, or both, in MoM’s config. See &quot;Job ncpus Limit Enforcement&quot; on page 397.</td>
<td>job-wide</td>
<td>MoM poll</td>
</tr>
</tbody>
</table>

5.15.3.4 Job Memory Limit Enforcement

You may wish to prevent jobs from swapping memory. To prevent this, you can set limits on the amount of memory a job can use. Then the job must request an amount of memory equal to or smaller than the amount of physical memory available.

PBS measures and enforces memory limits in two ways:

- On each host, by setting OS-level limits, using the limit system calls
- By periodically summing the usage recorded in the /proc entries.

Enforcement of mem is dependent on the following:

- Adding $enforce mem to the MoM’s config file
- The job requesting or inheriting a default value for mem
You can configure default \texttt{qsub} parameters in the \texttt{default_qsub_arguments} server attribute, or set memory defaults at the server or queue. See section 5.9.3, “Specifying Job Default Resources”, on page 305.

5.15.3.4.i Job Memory Limit Enforcement on UNIX

By default, memory limits are not enforced. To enforce \texttt{mem} resource usage, put \$\texttt{enforce mem} into MoM’s \texttt{config} file, and set defaults for \texttt{mem} so that each job inherits a value if it does not request it.

The \texttt{mem} resource can be enforced at both the job level and the vnode level. The job-wide limit is the smaller of a job-wide resource request and the sum of that for all chunks. The vnode-level limit is the sum for all chunks on that host.

Job-wide limits are enforced by MoM polling the working set size of all processes in the job’s session. Jobs that exceed their specified amount of physical memory are killed. A job may exceed its limit for the period between two polling cycles. See section 3.6.1, “Configuring MoM’s Polling Cycle”, on page 57.

Per-process limits are enforced by the operating system kernel. PBS calls the kernel call \texttt{setrlimit()} to set the limit for the top process (the shell), and any process started by the shell inherits those limits. PBS does not know whether the kernel kills a process for exceeding the limit.

If a user submits a job with a job limit, but not per-process limits (\texttt{qsub -l cput=10:00}) then PBS sets the per-process limit to the same value. If a user submits a job with both job and per-process limits, then the per-process limit is set to the lesser of the two values.

Example: a job is submitted with \texttt{qsub -lcput=10:00}

- There are two CPU-intensive processes which use 5:01 each. The job will be killed by PBS for exceeding the \texttt{cput} limit. 5:01 + 5:01 is greater than 10:00.
- There is one CPU-intensive process which uses 10:01. It is very likely that the kernel will detect it first.
- There is one process that uses 0:02 and another that uses 10:00. PBS may or may not catch it before the kernel does depending on exactly when the polling takes place.

If a job is submitted with a \texttt{pmem} limit, or without \texttt{pmem} but with a \texttt{mem} limit, PBS uses the \texttt{setrlimit(2)} call to set the limit. For most operating systems, \texttt{setrlimit()} is called with \texttt{RLIMIT_RSS} which limits the Resident Set (working set size). This is not a hard limit, but advice to the kernel. This process becomes a prime candidate to have memory pages reclaimed.

If \texttt{vmem} is specified and no single process exceeds that limit, but the total usage by all the processes in the job does, then PBS enforces the \texttt{vmem} limit, but not the \texttt{pvmem} limit, and logs a message. PBS uses MoM polling to enforce \texttt{vmem}.
Chapter 5  PBS Resources

The limit for \texttt{pmem} is enforced if the job specifies, or inherits a default value for, \texttt{pmem}. When \texttt{pmem} is enforced, the limit is set to the smaller of \texttt{mem} and \texttt{pmem}. Enforcement is done by the kernel, and applies to any single process in the job.

The limit for \texttt{pvmem} is enforced if the job specifies, or inherits a default value for, \texttt{pvmem}. When \texttt{pvmem} is enforced, the limit is set to the smaller of \texttt{vmem} and \texttt{pvmem}. Enforcement is done by the kernel, and applies to any single process in the job.

The following table shows which OS resource limits can be used by each operating system.

<table>
<thead>
<tr>
<th>OS</th>
<th>file</th>
<th>mem/pmем</th>
<th>vmem/pmем</th>
<th>cput/pcput</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>RLIMITFSIZE</td>
<td>RLIMIT_RSS</td>
<td>RLIMIT_DATA RLIMIT_STACK</td>
<td>RLIMIT_CPU</td>
</tr>
<tr>
<td>HP-UX</td>
<td>RLIMITFSIZE</td>
<td>RLIMIT_RSS</td>
<td>RLIMIT_AS</td>
<td>RLIMIT_CPU</td>
</tr>
<tr>
<td>Linux</td>
<td>RLIMITFSIZE</td>
<td>RLIMIT_RSS</td>
<td>RLIMIT_AS</td>
<td>RLIMIT_CPU</td>
</tr>
<tr>
<td>SunOS</td>
<td>RLIMITFSIZE</td>
<td>RLIMIT_DATA RLIMIT_STACK</td>
<td>RLIMIT_VMEM</td>
<td>RLIMIT_CPU</td>
</tr>
<tr>
<td>Super-UX</td>
<td>RLIMITFSIZE</td>
<td>RLIMIT_UMEM RLIMIT_DATA RLIMIT_STACK</td>
<td>ignored</td>
<td>RLIMIT_CPU</td>
</tr>
</tbody>
</table>

Note that \texttt{RLIMIT_RSS}, \texttt{RLIMIT_UMEM}, and \texttt{RLIMIT_VMEM} are not standardized (i.e. do not appear in the Open Group Base Specifications Issue 6).

5.15.3.4.ii  Sun Solaris-specific Memory Enforcement

Solaris does not support \texttt{RLIMIT_RSS}, but instead has \texttt{RLIMIT_DATA} and \texttt{RLIMIT_STACK}, which are hard limits. On Solaris or another Open Group standards-compliant OS, a \texttt{malloc()} call that exceeds the limit will return \texttt{NULL}. This behavior is different from other operating systems and may result in the program (such as a user’s application) receiving a \texttt{SIGSEGV} signal.
5.15.3.4.iii Memory Enforcement on cpusets

There should be no need to do so: either the vnode containing the memory in question has been allocated exclusively (in which case no other job will also be allocated this vnode, hence this memory) or the vnode is shareable (in which case using \texttt{mem\_exclusive} would prevent two CPU sets from sharing the memory). Essentially, PBS enforces the equivalent of \texttt{mem\_exclusive} by itself.

5.15.3.5 Job \texttt{ncpus} Limit Enforcement

Enforcement of the \texttt{ncpus} limit (number of CPUs used) is available on all platforms. The \texttt{ncpus} limit can be enforced using average CPU usage, burst CPU usage, or both. By default, enforcement of the \texttt{ncpus} limit is off. See “\texttt{\$enforce <limit>}” on page 271 of the PBS Professional Reference Guide.

5.15.3.5.i Average CPU Usage Enforcement

Each MoM enforces \texttt{cpuaverage} independently, per MoM, not per vnode. To enforce average CPU usage, put \texttt{\$enforce cpuaverage} in MoM’s \texttt{config} file. You can set the values of three variables to control how the average is enforced. These are shown in the following table.

<table>
<thead>
<tr>
<th>Table 5-14: Variables Used in Average CPU Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>cpuaverage</td>
</tr>
<tr>
<td>average_trialperiod</td>
</tr>
<tr>
<td>average_percent_over</td>
</tr>
<tr>
<td>average_cpusfactor</td>
</tr>
</tbody>
</table>
Enforcement of cpuaverage is based on the polled sum of CPU time for all processes in the job. The limit is checked each poll period. Enforcement begins after the job has had average_trialperiod seconds of walltime. Then, the job is killed if the following is true:

\[
\frac{\text{cput}}{\text{walltime}} > (\text{ncpus} \times \text{average_cpufactor} + \frac{\text{average_percent_over}}{100})
\]

5.15.3.5.ii CPU Burst Usage Enforcement

To enforce burst CPU usage, put $enforce \ cpuburst in MoM’s config file. You can set the values of four variables to control how the burst usage is enforced. These are shown in the following table.

### Table 5-15: Variables Used in CPU Burst

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpuburst</td>
<td>Boolean</td>
<td>If present (=True), MoM enforces ncpus when CPU burst usage exceeds specified limit.</td>
<td>False</td>
</tr>
<tr>
<td>delta_percent_over</td>
<td>integer</td>
<td>Modifies cpuburst. Percentage over limit to be allowed.</td>
<td>50</td>
</tr>
<tr>
<td>delta_cpufactor</td>
<td>float</td>
<td>Modifies cpuburst. ncpus limit is multiplied by this factor to produce actual limit.</td>
<td>1.5</td>
</tr>
<tr>
<td>delta_weightup</td>
<td>float</td>
<td>Modifies cpuburst. Weighting factor for smoothing burst usage when average is increasing.</td>
<td>0.4</td>
</tr>
<tr>
<td>delta_weightdown</td>
<td>float</td>
<td>Modifies cpuburst. Weighting factor for smoothing burst usage when average is decreasing.</td>
<td>0.1</td>
</tr>
</tbody>
</table>

MoM calculates an integer value called cpupercent each polling cycle. This is a moving weighted average of CPU usage for the cycle, given as the average percentage usage of one CPU. For example, a value of 50 means that during a certain period, the job used 50 percent of one CPU. A value of 300 means that during the period, the job used an average of three CPUs.

\[
\text{new\_percent} = \frac{\text{change\_in\_cpu\_time} \times 100}{\text{change\_in\_walltime}}
\]

\[
\text{weight} = \text{delta\_weight}[\text{up}\mid\text{down}] \times \frac{\text{walltime}}{\text{max\_poll\_period}}
\]

\[
\text{new\_cpupercent} = (\text{new\_percent} \times \text{weight}) + (\text{old\_cpupercent} \times (1 - \text{weight}))
\]
delta_weight_up is used if new_percent is higher than the old cpupercent value.

delta_weight_down is used if new_percent is lower than the old cpupercent value.

delta_weight_[up|down] controls the speed with which cpupercent changes. If
delta_weight_[up|down] is 0.0, the value for cpupercent does not change over time. If it is
1.0, cpupercent will take the value of new_percent for the poll period. In this case cpupercent
changes quickly.

However, cpupercent is controlled so that it stays at the greater of the average over the entire
run or ncpus*100.

max_poll_period is the maximum time between samples, set in MoM’s config file by
$max_check_poll, with a default of 120 seconds.

The job is killed if the following is true:

\[ new\_cpupercent \text{ > } (ncpus \times 100 \times \text{delta\_cpufactor}) + \text{delta\_percent\_over} \]

The following entries in MoM’s config file turn on enforcement of both average and burst
with the default values:

$enforce\ cpuaverage
$enforce\ cpuburst
$enforce\ delta\_percent\_over\ 50
$enforce\ delta\_cpufactor\ 1.05
$enforce\ delta\_weightup\ 0.4
$enforce\ delta\_weightdown\ 0.1
$enforce\ average\_percent\_over\ 50
$enforce\ average\_cpufactor\ 1.025
$enforce\ average\_trialperiod\ 120

The cpuburst and cpuaverage information show up in MoM’s log file, whether or not they
have been configured in mom_priv/config. This is so a site can test different parameters
for cpuburst/cpuaverage before enabling enforcement. You can see the effect of any
change to the parameters on your job mix before "going live".

Note that if the job creates a child process whose usage is not tracked by MoM during its life-
time, CPU usage can appear to jump dramatically when the child process exits. This is
because the CPU time for the child process is assigned to its parent when the child process
exits. MoM may see a big jump in cpupercent, and kill the job.

5.15.3.5.iii Job Memory Limit Restrictions

Enforcement of mem resource usage is available on all UNIX platforms, but not Windows.
5.15.3.6 Changing Job Limits

The `qalter` command is used to change job limits, with these restrictions:

- A non-privileged user may only lower the limits for job resources
- A Manager or Operator may lower or raise requested resource limits, except for per-process limits such as `pcput` and `pmem`, because these are set when the process starts, and enforced by the kernel.
- When you lengthen the `walltime` of a running job, make sure that the new `walltime` will not interfere with any existing reservations etc.

See “`qalter`” on page 127 of the PBS Professional Reference Guide.

5.16 Where Resource Information Is Kept

Definitions and values for PBS resources are kept in the following files, attributes, and parameters. Attributes specifying resource limits are not listed here. They are listed in section 5.15.1.8, “Resource Usage Limit Attributes for Server and Queues”, on page 376 and section 5.15.1.15, “Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1”, on page 388.

5.16.1 Files

`PBS_HOME/server_priv/resourcedef`
Contains definitions of custom resources. Format:

```
<resource name> [type=<type>] [flag=<flags>]
```

Example:

```
LocalScratch type=long flag=h
FloatLicense type=long
SharedScratch type=long
```


`PBS_HOME/sched_priv/sched_config`
resources: line

In order for scheduler to be able to schedule using a resource, the resource must be listed in the `resources: line`. Format:
resources: “<resource name>, [<resource name>, ...]”
Example:
  resources: “ncpus, mem, arch, [...], LocalScratch, FloatLicense, SharedScratch”

The only exception is host-level Boolean resources, which do not need to appear in the resources: line.

server_dyn_res: line
Each dynamic server resource must be listed in its own
server_dyn_res: line. Format:
server_dyn_res: “<resource name> !<path to script/command>”
Example:
  server_dyn_res: “SharedScratch !/usr/local/bin/serverdynscratch.pl”

mom_resources: line
Dynamic host resources must be listed in the mom_resources: line. Format:
mom_resources: “<resource name>”
Example:
  mom_resources: “LocalScratch”

PBS_HOME/mom_priv/config
Contains MoM configuration parameters and any local resources. Format:
<resource name> !<path to script/command>
Example:
  LocalScratch !/usr/local/bin/localscratch.pl
See “MoM Parameters” on page 267 of the PBS Professional Reference Guide.

5.16.2 MoM Configuration Parameters

$cputmult <factor>
This sets a factor used to adjust CPU time used by each job. This allows adjustment of time charged and limits enforced where jobs run on a system with different CPU performance. If MoM’s system is faster than the reference system, set factor to a decimal value greater than 1.0. For example:
  $cputmult 1.5
If MoM’s system is slower, set factor to a value between 1.0 and 0.0. For example:

```bash
$cputmult 0.75
```

`$wallmult <factor>`
Each job’s `walltime` usage is multiplied by this factor. For example:

```bash
$wallmult 1.5
```

### 5.16.3 Server Attributes

**default_chunk**
The list of resources which will be inserted into each chunk of a job’s select specification if the corresponding resource is not specified by the user. This provides a means for a site to be sure a given resource is properly accounted for even if not specified by the user.

*Format:* `String`

*Form:* `default_chunk.<resource>=<value>,default_chunk.<resource>=<value>,...`

*Default:* None

**default_qsub_arguments**
Arguments that are automatically added to the `qsub` command. Any valid arguments to `qsub` command, such as job attributes. Setting a job attribute via `default_qsub_arguments` sets that attribute for each job which does not explicitly override it. See `qsub(1B)`. Settable by the administrator via the `qmgr` command. Overrides standard defaults. Overridden by arguments given on the command line and in script directives.

*Example:*

```bash
Qmgr: set server default_qsub_arguments="-r y -N MyJob"
```

*Form:* `String`

*Default:* None

**resources_available.<resource name>**
The list of available resources and their values defined on the server. Each resource is listed on a separate line.

*Format:* `String`

*Form:* `resources_available.<resource>=<value>`

*Default:* None
resources_default.<resource name>
The list of default resource values that are set as limits for jobs in this complex when a) the job does not specify a limit, and b) there is no queue default.
Format: String.
Form: resources_default.<resource_name>=value[,...]
Default: None

resources_assigned.<resource name>
The total of each type of consumable resource allocated to running jobs and started reservations in this complex. Read-only.
Format: String.
Form: resources_assigned.<res>=<val>[,resources_assigned.<res>=<val>,...]
Default: None

5.16.4 Reservation Attributes

Resource_List.<resource name>
The list of resources allocated to the reservation. Jobs running in the reservation cannot use in aggregate more than the specified amount of a resource.
Format: String
Form: Resource_List.<res>=<val>, Resource_List.<res>=<val>, ...
Default: None

5.16.5 Queue Attributes

default_chunk.<resource name>
The list of resources which will be inserted into each chunk of a job’s select specification if the corresponding resource is not specified by the user. This provides a means for a site to be sure a given resource is properly accounted for even if not specified by the user. Applies only to execution queues.
Format: String.
Form: default_chunk.<resource>=<value>, default_chunk.<resource>=<value>, ...
Default: None
resources_default.<resource name>
The list of default resource values which are set as limits for a job residing in this queue and for which the job did not specify a limit. If not set, the default limit for a job is determined by the first of the following attributes which is set: server’s resources_default, queue’s resources_max, server’s resources_max. If none of these is set, the job gets unlimited resource usage
Format: String.
Form: resources_default.<resource name>=<value>, resources_default.<resource name>=<value>, ...
Default: None

resources_assigned.<resource name>
The total for each kind of consumable resource allocated to jobs running from this queue. Read-only.
Format: String.
Form: resources_assigned.<resource name>=<value><newline>resources_assigned.<resource name>=<value><newline>...
Default: None

resources_available.<resource name>
The list of resources and amounts available to jobs running in this queue. The sum of the resource of each type used by all jobs running from this queue cannot exceed the total amount listed here. See “qmgr” on page 150 of the PBS Professional Reference Guide.
Format: String.
Form: resources_available.<resource name>=<value><newline>resources_available.<resource name>=<value><newline>...
Default: None

5.16.6 Vnode Attributes

resources_available.<resource name>
The list of resources and the amounts available on this vnode. If not explicitly set, the amount shown is that reported by the pbs_mom running on the vnode. If a resource value is explicitly set, that value is retained across restarts.
Format: String.
Form: resources_available.<resource name>=<value>, resources_available.<resource name>=<value>, ...
Default: None
sharing
Specifies whether more than one job at a time can use the resources of the vnode or the vnode’s host. Either (1) the vnode or host is allocated exclusively to one job, or (2) the vnode’s or host’s unused resources are available to other jobs. Can be set using `pbs_mom -s insert` only. Behavior is determined by a combination of the `sharing` attribute and a job’s placement directive. See “sharing” on page 371 of the PBS Professional Reference Guide.

pcpus
The number of physical CPUs on the vnode. This is set to the number of CPUs available when MoM starts. For a multiple-vnode MoM, only the natural vnode has `pcpus`.

Format: `Integer`

Default: Number of CPUs on startup

resources_assigned.<resource name>
The total amount of each consumable resource allocated to jobs and started reservations running on this vnode. Applies only to execution queues. Read-only.

Format: `String`

Form:
```
resources_assigned.<resource>=<value>[,resources_assigned.<resource>=<value>]
```

Default: None

5.16.7  Job Attributes

Resource_List.<resource name>
The list of resources required by the job. List is a set of `<name>=<value>` strings. The meaning of name and value is dependent upon defined resources. Each value establishes the limit of usage of that resource. If not set, the value for a resource may be determined by a queue or server default established by the administrator. See section 5.9.2, “Resources Requested by Job”, on page 305.

Format: `String`

Form: `Resource_List.<res>=<value>, Resource_List.<res>=<value>, ...`

Default: None

resources_used.<resource name>
The amount of each resource actually used by the job. Read-only.

Format: `String`
Form: List of `<name>=<value>` pairs: `resources_used.<res>=<val>`, `resources_used.<res>=<val>`

5.17 Viewing Resource Information

You can see attribute values of resources for the server, queues, and vnodes using the `qmgr` or `pbsnodes` commands. The value in the server, queue, or vnode `resources_assigned` attribute is the amount explicitly requested by jobs and, at the server and vnodes, started reservations.

You can see job attribute values using the `qstat` command. The value in the job’s `Resource_List` attribute is the amount explicitly requested by the job. See section 5.9.2, “Resources Requested by Job”, on page 305.

The following table summarizes how to find resource information:

<table>
<thead>
<tr>
<th>Location</th>
<th>Item to View</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td><code>default_chunk</code>, <code>default_qsub_arguments</code>, <code>resources_available</code>, <code>resources_assigned</code>, <code>resources_default</code></td>
<td><code>qmgr</code>, <code>qstat</code>, <code>pbsnodes</code></td>
</tr>
<tr>
<td></td>
<td><code>resources_def_file</code></td>
<td>Favorite editor or viewer</td>
</tr>
<tr>
<td>scheduler</td>
<td><code>sched_config_file</code></td>
<td>Favorite editor or viewer</td>
</tr>
<tr>
<td>queues</td>
<td><code>default_chunk</code>, <code>resources_available</code>, <code>resources_assigned</code>, <code>resources_default</code></td>
<td><code>qmgr</code>, <code>qstat</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MoM and vnodes</td>
<td><code>resources_available</code>, <code>sharing</code>, <code>pcpus</code>, <code>resources_assigned</code></td>
<td><code>qmgr</code>, <code>pbsnodes</code></td>
</tr>
<tr>
<td></td>
<td><code>mom_config_file</code></td>
<td>Favorite editor or viewer</td>
</tr>
<tr>
<td>job</td>
<td><code>Resource_List</code></td>
<td><code>qstat</code></td>
</tr>
<tr>
<td>reservation</td>
<td><code>Resource_List</code></td>
<td><code>pbs_rstat -f</code></td>
</tr>
</tbody>
</table>
Every consumable resource, for example mem, can appear in four PBS attributes. These attributes are used in the following elements of PBS:

Table 5-17: Values Associated with Consumable Resources

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Vnode</th>
<th>Queue</th>
<th>Server</th>
<th>Accounting Log</th>
<th>Job</th>
<th>Scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>resources_available</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>resources_assigned</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>resources_used</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource_List</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

5.17.1 Resource Information in Accounting Logs

You can see accounting values in the accounting log file. The accounting log S record is written at the start of a job, and the E record is written at the end of a job. The accounting log B record is written at the beginning of a reservation.

Each consumable resource allocated to or taken up by a job is reported separately in a resource_assigned accounting entry in the job’s E and S records. The resource_assigned entry is not a job attribute; it is simply an entry in the accounting log.

Consumable job resources actually used by the job are recorded in the job’s resources_used attribute, and are reported in the accounting log.

The value reported in the resource_assigned accounting entry is the amount assigned to a job or that a job prevents other jobs from using, which is different from the amount the job requested and used. For example, if a job requests one CPU on an Altix that has four
Chapter 5  

PBS Resources

CPUs per blade/vnode and that vnode is allocated exclusively to the job, even though the job requested one CPU, it is assigned all 4 CPUs. In this example, resource_assigned reports 4 CPUs, and resources_used reports 1 CPU.

Resources requested for a job are recorded in the job’s Resource_List attribute, and reported in the accounting log E and S records for the job.

Resources requested for a reservation are recorded in the reservation’s Resource_List attribute, and reported in the accounting log B record for the reservation.

5.17.2 Resource Information in Daemon Logs

At the end of each job, the server logs the values in the job’s resources_used attribute, at event class 0x0010.

Upon startup, MoM logs the number of CPUs reported by the OS, at event class 0x0002.

At the end of each job, the MoM logs cput and mem used by each job, and cput used by each job task, at event class 0x0100.

5.17.3 Finding Current Value

You can find the current value of a resource by subtracting the amount being used from the amount that is defined.

Use the qstat -Bf command, and grep for resources_available.<resource> and resources_used.<resource>. To find the current amount not being used, subtract resources_used.<resource> from resources_available.<resource>.

5.17.4 Restrictions on Viewing Resources

- Dynamic resources shown in qstat do not display the current value, they display the most recent retrieval. Dynamic resources have no resources_available.<resource> representation anywhere in PBS.
- Local static host-level resources cannot be viewed via qstat or managed via qmgr.

5.18 Resource Recommendations and Caveats

- It is not recommended to set the value for resources_available.ncpus. The exception is when you want to oversubscribe CPUs. See section 9.4.5.1.iii, “How To Share CPUs”, on page 740.
- It is not recommended to change the value of ncpus at vnodes on a multi-vnoded
• If you want to limit how many jobs are run, or how much of each resource is used, use the new limits. See section 5.15, “Managing Resource Usage”, on page 366.

• It is not recommended to create local host-level resources by defining them in the MoM configuration file.

• On the Altix, do not set the values for mem, vmem or ncpus on the natural vnode. If any of these resources has been explicitly set to a non-zero value on the natural vnode, set resources_available.ncpus, resources_available.mem and resources_available.vmem to zero on each natural vnode:

• Do not attempt to set values for resources_available.<resource> for dynamic resources.

• Externally-managed licenses may not be available when PBS thinks they are. PBS doesn't actually check out externally-managed licenses; the application being run inside the job's session does that. Between the time that the scheduler queries for licenses, and the time the application checks them out, another application may take the licenses. In addition, some applications request varying amounts of tokens during a job run.

• Jobs may be placed on different vnodes from those where they would have run in earlier versions of PBS. This is because a job’s resource request will no longer match the same resources on the server, queues and vnodes.

• While users cannot request custom resources that are created with the r flag, jobs can inherit these as defaults from the server or queue resources_default.<resource> attribute.

• A qsub or pbs_rsub hook does not have resources inherited from the server or queue resources_default or default_chunk as an input argument.

• Resources assigned from the default_qsub_arguments server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag, whether that resource was requested by the user or came from default_qsub_arguments. Be aware that creating custom resources with permission flags and then using these in the default_qsub_arguments server attribute can cause jobs to be rejected. See section 5.14.2.6, “Resource Permission Flags”, on page 328.

• Numeric dynamic resources cannot have the q or n flags set. This would cause these resources to be underused. These resources are tracked automatically by scheduler.

• The behavior of several command-line interfaces is dependent on resource permission flags. These interfaces are those which view or request resources or modify resource requests:

  pbsnodes
  Users cannot view restricted host-level custom resources.

  pbs_rstat
  Users cannot view restricted reservation resources.
Chapter 5  PBS Resources

pbs_rsub
Users cannot request restricted custom resources for reservations.

qalter
Users cannot alter a restricted resource.

qmgr
Users cannot print or list a restricted resource.

qselect
Users cannot specify restricted resources via -l Resource_List.

qsub
Users cannot request a restricted resource.

qstat
Users cannot view a restricted resource.

• Do not set values for any resources, except those such as shared scratch space or floating licenses, at the server or a queue, because the scheduler will not allocate more than the specified value. For example, if you set resources_available.walltime at the server to 10:00:00 and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.

• If a job is submitted without a request for a particular resource, and no defaults for that resource are set at the server or queue, and either the server or queue has resources_max.<resource> set, the job inherits that maximum value. If the queue has resources_max.<resource> set, the job inherits the queue value, and if not, the job inherits the server value.

• Do not create custom resources with the same names or prefixes that PBS uses when creating custom resources for specific systems. See “Custom Cray Resources” on page 308 of the PBS Professional Reference Guide.

• Do not set resources_available.place for a vnode.

• Using dynamic host-level resources can slow the scheduler down, because the scheduler must wait for each resource-query script to run.

• On the natural vnode, all values for resources_available.<resource> should be zero (0), unless the resource is being shared among other vnodes via indirection.

• Default qsub arguments and server and queue defaults are applied to jobs at a coarse level. Each job is examined to see whether it requests a select and a place. This means that if you specify a default placement, such as excl, with -lplace=excl, and the user specifies an arrangement, such as pack, with -lplace=pack, the result is that the job ends up with -lplace=pack, NOT -lplace=pack:excl. The same is true for select, if you specify a default of -lselect=2:ncpus=1, and the user specifies -
lselect=mem=2GB, the job ends up with \texttt{!lselect=mem=2GB}. 
Chapter 5  PBS Resources

-------------------------------

412  PBS Professional 12.2 Administrator’s Guide
Chapter 6

Hooks

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches or workarounds, and extends the capabilities of PBS, without the need to modify source code.

This chapter describes how hooks can be used, how they work, the interface to hooks provided by the pbs module, how to create and deploy hooks, and how to get information about hooks.

Please read the entire chapter, and the “Special Notes (Hooks)” section of the release notes, before writing any hooks.

6.1 Chapter Contents

6.1 Chapter Contents .......................................................... 413
6.2 Introduction to Hooks ..................................................... 416
6.3 Terminology ..................................................................... 417
6.4 Prerequisites and Requirements for Hooks ......................... 418
6.5 Updating Hooks and Custom Resources ............................. 419
6.6 Simple How-to for Writing Hooks ................................. 420
   6.6.1 Writing Hooks: Basic Hook Structure ...................... 420
   6.6.2 Example of Simple Hook ........................................ 422
   6.6.3 Creating and Importing Your Hook ......................... 422
   6.6.4 Setting Attributes for Your Hook .............................. 423
### Chapter 6  
**Hooks**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6.5</td>
<td>Simplified View of Hook Objects</td>
<td>424</td>
</tr>
<tr>
<td>6.7</td>
<td>Examples of Using Hooks</td>
<td>426</td>
</tr>
<tr>
<td>6.7.1</td>
<td>Routing Jobs Using Hooks</td>
<td>426</td>
</tr>
<tr>
<td>6.7.2</td>
<td>Managing Job Resource Requests</td>
<td>426</td>
</tr>
<tr>
<td>6.7.3</td>
<td>Managing Access to Resources for Users and Jobs</td>
<td>426</td>
</tr>
<tr>
<td>6.7.4</td>
<td>Ensuring Efficient Use of Resources</td>
<td>427</td>
</tr>
<tr>
<td>6.7.5</td>
<td>Ensuring that Jobs Run Properly</td>
<td>427</td>
</tr>
<tr>
<td>6.7.6</td>
<td>Controlling Interactive Jobs</td>
<td>427</td>
</tr>
<tr>
<td>6.7.7</td>
<td>Communicating Information to Users</td>
<td>427</td>
</tr>
<tr>
<td>6.7.8</td>
<td>Converting Requests to Usable Format</td>
<td>427</td>
</tr>
<tr>
<td>6.7.9</td>
<td>Helping Schedule Jobs</td>
<td>428</td>
</tr>
<tr>
<td>6.7.10</td>
<td>Managing User Activity</td>
<td>428</td>
</tr>
<tr>
<td>6.7.11</td>
<td>Enabling Accounting and Validation</td>
<td>428</td>
</tr>
<tr>
<td>6.7.12</td>
<td>Enforcing Security</td>
<td>429</td>
</tr>
<tr>
<td>6.7.13</td>
<td>Provisioning Vnodes</td>
<td>429</td>
</tr>
<tr>
<td>6.7.14</td>
<td>Managing Job Output</td>
<td>429</td>
</tr>
<tr>
<td>6.7.15</td>
<td>Allocation Management</td>
<td>429</td>
</tr>
<tr>
<td>6.7.16</td>
<td>Helping Manage Job Execution</td>
<td>429</td>
</tr>
<tr>
<td>6.8</td>
<td>When, as Whom, and Where Hooks Run</td>
<td>430</td>
</tr>
<tr>
<td>6.8.1</td>
<td>Overview of When Hooks Run</td>
<td>430</td>
</tr>
<tr>
<td>6.8.2</td>
<td>Execution Event Hook Triggers in Lifecycle of Job</td>
<td>433</td>
</tr>
<tr>
<td>6.8.3</td>
<td>Account Under Which Hooks Run</td>
<td>434</td>
</tr>
<tr>
<td>6.8.4</td>
<td>Where Hooks Run</td>
<td>434</td>
</tr>
<tr>
<td>6.9</td>
<td>What Hooks Do</td>
<td>434</td>
</tr>
<tr>
<td>6.9.1</td>
<td>Accepting or Rejecting Actions</td>
<td>434</td>
</tr>
<tr>
<td>6.9.2</td>
<td>Modifying Job Submission (qsub)</td>
<td>435</td>
</tr>
<tr>
<td>6.9.3</td>
<td>Modifying Job Change (qalter)</td>
<td>435</td>
</tr>
<tr>
<td>6.9.4</td>
<td>Modifying Job Move (qmove)</td>
<td>436</td>
</tr>
<tr>
<td>6.9.5</td>
<td>Modifying Reservation Creation (pbs_rsub)</td>
<td>436</td>
</tr>
<tr>
<td>6.9.6</td>
<td>Changes Before Job is Sent to MoM (qrun)</td>
<td>436</td>
</tr>
<tr>
<td>6.9.7</td>
<td>Changes When Job is Received by MoM</td>
<td>437</td>
</tr>
<tr>
<td>6.9.8</td>
<td>Changes Before Job is Executed</td>
<td>437</td>
</tr>
<tr>
<td>6.9.9</td>
<td>Changes After Job is Executed</td>
<td>438</td>
</tr>
<tr>
<td>6.9.10</td>
<td>Changes Before Job is Killed</td>
<td>438</td>
</tr>
<tr>
<td>6.9.11</td>
<td>Changes After Job Finishes or is Killed</td>
<td>438</td>
</tr>
<tr>
<td>6.9.12</td>
<td>Periodic Events at Execution Hosts</td>
<td>438</td>
</tr>
<tr>
<td>6.9.13</td>
<td>Failover</td>
<td>439</td>
</tr>
<tr>
<td>6.10</td>
<td>Creating Hooks</td>
<td>439</td>
</tr>
<tr>
<td>6.10.1</td>
<td>Introduction to Creating and Deleting Hooks</td>
<td>439</td>
</tr>
<tr>
<td>6.10.2</td>
<td>Creating Empty Hooks</td>
<td>441</td>
</tr>
<tr>
<td>6.10.3</td>
<td>Deleting Hooks</td>
<td>441</td>
</tr>
<tr>
<td>6.10.4</td>
<td>Setting Hook Trigger Events</td>
<td>441</td>
</tr>
</tbody>
</table>
6.10.5 Importing Hooks ................................................. 443
6.10.6 Exporting Hooks .............................................. 444

6.11 Configuring Hooks .............................................. 446
6.11.1 Hook Attributes ............................................. 446
6.11.2 Setting Hook Trigger Events ................................. 448
6.11.3 Enabling and Disabling Hooks ............................... 449
6.11.4 Setting the Relative Order of Hook Execution ............ 449
6.11.5 Setting Hook Timeout ....................................... 450
6.11.6 Setting Hook Frequency ..................................... 451
6.11.7 Setting Hook User Account .................................. 451

6.12 Viewing Hook Information ..................................... 452
6.12.1 Listing Hooks .................................................. 452
6.12.2 Viewing Hook Contents ..................................... 452
6.12.3 Printing Hook Creation Commands .......................... 452
6.12.4 Re-creating Hooks ............................................ 453

6.13 Interface to Hooks .............................................. 454
6.13.1 The pbs Module .............................................. 454
6.13.2 What Hooks Can Access and Do ............................. 454
6.13.3 What Hooks Cannot Access or Do ............................ 455
6.13.4 What Hooks Should Not Do .................................. 455
6.13.5 Using Attributes and Resources in Hooks .................. 456
6.13.6 Using select and place in Hooks .............................. 472
6.13.7 PBS Interface Objects ....................................... 473
6.13.8 Constant Objects ............................................ 485
6.13.9 Event Objects ................................................ 492
6.13.10 Server Objects .............................................. 515
6.13.11 Queue Objects .............................................. 518
6.13.12 Job Objects ................................................ 519
6.13.13 exec_vnode Object .......................................... 524
6.13.14 Chunk Objects .............................................. 526
6.13.15 Reservation Objects ........................................ 526
6.13.16 Vnode Objects .............................................. 528
6.13.17 Global Attribute Creation Methods ......................... 529
6.13.18 Other Global Creation Methods ............................. 534

6.14 Hook Examples .................................................... 536

6.15 Advice and Caveats .............................................. 548
6.15.1 Rules for Hook Access and Behavior ......................... 548
6.15.2 Referencing the Object You Want ............................ 549
6.15.3 Adding and Deleting Vnodes ................................. 549
6.15.4 Printing And Logging Messages .............................. 550
6.15.5 Capturing Return Code ...................................... 550
6.15.6 Recommended Hook Script Structure ......................... 551
6.2 Introduction to Hooks

A hook is a block of Python code that PBS executes at certain events, for example, when a job is queued. As long as the Python code conforms to the rules we describe, you can have it do whatever you want. Hooks can be roughly grouped into those that run before a job is received by the execution host, those that run after a job is received by the execution host, and hooks that aren’t directly related to a specific job.
Hooks

Hooks that run before a job is received by an execution host (pre-execution event hooks):

- Queueing a job
- Modifying a job (can also run after job is received by execution host)
- Moving a job
- Just before a job is sent to an execution host

Hooks that run after a job is received by an execution host (execution event hooks):

- When a job is received by an execution host
- Just before executing a job
- Just after executing a job
- Just before killing a job
- Just after killing a job

Hooks that are not directly related to a specific job (non-job event hooks):

- Submitting a PBS reservation
- Provisioning a vnode
- Periodically on all execution hosts

Each hook can accept (allow) or reject (prevent) the action that triggers it. The hook can modify the input parameters given for the action. The hook can also make calls to functions external to PBS. PBS provides an interface for use by hooks. This interface allows hooks to read and/or modify things such as job, server, vnode, and queue attributes, and the event that triggered the hook.

6.3 Terminology

Action

A PBS operation or state transition. The actions that hooks can affect are submitting a job, altering a job, sending a job to the execution host, making a reservation, moving a job to another queue, and receiving the job at the execution host, starting the job, ending the job, killing the job, and finishing final cleanup of a job. There is also a periodic action at all execution hosts.

Accept an action

The hook allows the action to take place.

Execution event hook

A hook that runs at an execution host. These hooks run after a job is received by MoM. Execution event hooks are for receiving the job at the execution host,
starting the job, ending the job, killing the job, and finishing final cleanup of a job. Execution events have names prefixed with “execjob_”.

Non-job event hook
A hook that is not directly related to a specific job. Non-job event hooks are periodic hooks, provisioning hooks, and reservation creation hooks.

pbs module
The pbs module is an interface to PBS and the hook environment. The interface is made up of Python objects, which have members and methods. You can operate on these objects using Python code.

Reject an action
The hook prevents the action from taking place.

Pre-execution event hook
A hook that runs before the job is accepted by MoM. These hooks do not run on execution hosts. Pre-execution event hooks are for job submission, moving a job, altering a job, or just before sending a job to an execution host.

6.4 Prerequisites and Requirements for Hooks

• To create a hook under UNIX/Linux, you must be logged into the primary or secondary server host as root. You must create any hooks at the primary or secondary server host.

• To create a hook under Windows, you must use the installation account. For domained environments, the installation account must be a local account that is a member of the local Administrators group on the local computer. For standalone environments, the installation account must be a local account that is a member of the local Administrators group on the local computer.

• On Windows 7 and later with UAC enabled, if you will use the cmd prompt to operate on hooks, or for any privileged command such as qmgr, you must run the cmd prompt with option Run as Administrator.

• When creating hooks, make sure that each execution host where execution or periodic hooks should run has the $reject_root_scripts MoM parameter set to False. The default for this parameter is False.

A normal, non-privileged, user cannot circumvent, disable, add, delete, or modify hooks or the environment in which the hooks are run.
6.5 Updating Hooks and Custom Resources

6.5.1 Read Release Notes

Make sure that you read the “Special Notes (Hooks)” section of the PBS Professional release notes before writing any hooks.

6.5.2 Hook Updates are Asynchronous for MoMs

Updates to hooks are asynchronous with respect to jobs. When you create or modify an execution event hook, PBS must propagate the changes out to all the MoMs. On a complex with a few MoMs, the time required for updates to propagate is negligible, but where there are many MoMs, this process can take up to ten minutes or more.

During the update, some jobs may run on updated MoMs while others run on MoMs that are not yet updated. A multi-host job that started running before the update may find itself running on some MoMs that are updated and some that are not. In addition, a multi-host job that starts during the update may start on updated and non-updated MoMs.

When a job triggers a hook, the hook that runs is the current hook, not the hook that was there when the job started. If you change, delete, or add a hook while a job is running, and the job subsequently triggers the hook, that job will encounter whatever changes have propagated to the MoM.

If you add many MoMs all at once, PBS will need time to propagate hooks to the new MoMs.

6.5.3 Hooks that Depend on New Custom Resources

When you add or change a custom resource on a large complex, it can take up to ten minutes or more for PBS to propagate the changes to all MoMs. On small complexes, the time required is negligible. If a custom resource is created or changed, wait at least ten minutes before creating any hooks that use the resource.

Custom resources are propagated under the following circumstances:

- When you add a resource to the resourcedef file
- When you install PBS on a multi-vnoded machine
- When you add MoMs, resources are propagated to those MoMs
6.5.3.1 Caveats for Hooks that Use Custom Resources

If a hook tries to use a resource that is not yet propagated, this will cause an exception, which if unhandled, may delete the job. Write your hooks so that they trap exceptions and deal gracefully with the job. For example, you can use pbs.event().job.rerun().

6.6 Simple How-to for Writing Hooks

We will go into the details of what goes into a hook later in the chapter, but here we show the basics of how to create a hook. In order to create a hook, you must be logged into the server host as root. Steps for creating a hook:

1. Write the hook script
2. Create an empty hook via qmgr
3. Set the attributes of the hook so that it triggers when you want, etc
4. Import the hook script into the empty hook. You do not need to restart the MoM.

6.6.1 Writing Hooks: Basic Hook Structure

- Import the pbs and sys modules:
  
  ```python
  import pbs
  import sys
  ```

- Use the try...except construction, where you test for conditions in the try block, and accept or reject the event:
  
  ```python
  try:
  ...
  except:
  ```

  Consider either rerunning the job or deleting the job inside the except: block.

- Treat the SystemExit exception as a normal occurrence, and pass if it occurs:
  
  ```python
  except SystemExit:
  pass
  ```

- Reject the event, or rerun or delete the job, if any other exception occurs:
  
  ```python
  except:
  pbs.event().reject("%s hook failed with %s")
  ```

- If the requestor is the scheduler, and where appropriate, the server or MoM, allow the
action to take place:

```python
if pbs.event().requestor in ['PBS_Server', 'Scheduler', 'pbs_mom']:
    pbs.event().accept()
```

The following code fragment is a basic hook skeleton:

```python
import pbs
import sys

e = pbs.event()
j = e.job
try:
    if e.requestor in ['Scheduler']:
        e.accept()

except SystemExit:
    pass

except:
    j.rerun()
    e.reject("%s hook failed with %s. Please contact Admin" % (e.hook_name, sys.exc_info()[2]))
```
6.6.2 Example of Simple Hook

Example 6-1: Redirecting newly-submitted jobs:

If a job is submitted to a queue other than workq, move it to workq

```python
import pbs
import sys

try:
    # Get the hook event information and parameters
    # This will be for the 'queuejob' event type.
    e = pbs.event()

    # Ignore requests from scheduler or server
    if e.requestor in ["PBS_Server", "Scheduler"]:
        e.accept()

    # Get the information for the job being queued
    j = e.job
    if j.queue in ["long", "short"]:
        j.queue = pbs.server().queue("workq")
    # accept the event
    e.accept()
except SystemExit:
    pass
except:
    e.reject("Failed to route job to queue workq")
```

6.6.3 Creating and Importing Your Hook

When you “create a hook” using qmgr, you’re telling PBS that you want it to make you an empty hook object that has no characteristics other than a name. When you “import a hook” using qmgr, you’re telling PBS which Python script to run when the hook is triggered.

Format for creating a hook:

```
Qmgr: create hook <hook name>
```
Simple format for importing a hook:

```
Qmgr: import hook <hook name> application/x-python <content-encoding> <input_file>
```

This uses the script named `<input_file>` as the contents of your hook.

- The `<input_file>` must be encoded with `<content-encoding>`.
- The allowed values for `<content-encoding>` are “default” (7 bit) and “base64”.
- `<input_file>` must be locally accessible to both `qmgr` and the batch server.
- A relative path in `<input_file>` is relative to the directory where `qmgr` was executed.
- If your hook already has a content script, then that is overwritten by this import call.
- If the name of `<input_file>` contains spaces, as are used in Windows filenames, then `<input_file>` must be quoted.

### 6.6.4 Setting Attributes for Your Hook

Hooks have attributes that control their behavior, such as which events trigger the hook, the time to allow the hook to execute, etc. The only attribute you have to set for a simple hook is the event(s) that will trigger the hook. Choose your hook type according to the event you want, by looking in section 6.8, “When, as Whom, and Where Hooks Run”, on page 430.

Format for setting the hook event(s):

```
Qmgr: set hook <hook name> event = <event name>
Qmgr: set hook <hook name> event = “[<event name>, <event name>]”
```

For more details on setting hook trigger events, see section 6.11.2, “Setting Hook Trigger Events”, on page 448.

You can set the rest of the hook’s attributes if you wish. To set a hook attribute:

```
Qmgr: set hook <hook name> <attribute> = <value>
```

For a list of all the hook attributes, see section 6.11.2, “List of Hook Attributes”, on page 446.
6.6.5 Simplified View of Hook Objects

Figure 6-1 shows a simplified view of event, job, server, vnode, queue, resv, and vchunk objects. All hook objects have the methods listed in “global methods”. Each object also has its own members and methods, as shown. We show event objects in more detail in figure 6-2.

Figure 6-1: Simplified view of hook objects
Figure 6-2 shows a simplified view of hook event objects. All hook events have the members and methods listed in “event (all)”. Each type of event also has its own members and/or methods. For example, movejob events have a job member and a src_queue member, in addition to the type, hook_name, requestor, requestor_host, hook_type, and hook_alarm members, and the accept(), get_local_nodename(), logjobmsg(), logmsg(), and reject() methods shared by all events. For a complete description of event objects, see section 6.13.9, “Event Objects”, on page 492.
6.7 Examples of Using Hooks

6.7.1 Routing Jobs Using Hooks

- Route jobs into specific queues or between queues:
  - Automatically route interactive jobs into a particular execution queue
  - Move a job to another queue; for example, if project allocation is used up, move job to “background” queue
- Reject job submissions that do not specify a valid queue, printing an error message explaining the problem
- Enable project-based ACLs for queues to make sure the appropriate job runs in the correct queue

6.7.2 Managing Job Resource Requests

- Reject improperly specified jobs:
  - Reject jobs which do not specify walltime
  - Reject jobs that request a number of processors that is not a multiple of 8
  - Reject jobs requesting a specific queue, but not requesting memory
  - Reject jobs whose processors per node is not specified or is not numeric
- Modify job resource requests:
  - Apply default memory limit to jobs that request a specific queue
  - Check on requested CPU and memory and modify these or supply them if missing
  - Adjust for the fact that users ask for 2GB on an Altix that has 2GB physical memory, but only 1.8 GB available memory, by changing the memory request to 1.8GB

6.7.3 Managing Access to Resources for Users and Jobs

- Compensate for dissimilar system capabilities; for example, allow users to use more CPUs only if they use old, slow machines.
- Limit reservations submitted by users to a maximum amount of resources and walltime, but do not limit reservations submitted by PBS administrators.
6.7.4 Ensuring Efficient Use of Resources

- Reject parallel jobs for some queues.
- Set default properties, for example, if “myri” is not set, set it to “False” to ensure Myrinet is used only for Myrinet jobs.

6.7.5 Ensuring that Jobs Run Properly

- Make sure that jobs, or all jobs in a queue, request exclusive access (-l place=excl).
- Reject multi-host jobs, restricting all jobs to a single Altix.
- Put a hold on the job if there isn’t enough scratch space when the job is submitted.
- Reject jobs that could cause problems, based on the user and type of job that have caused previous problems. For example, if Bill's Abaqus jobs crash the system, reject new Abaqus jobs from Bill.
- Validate an input deck before the job is submitted.
- Modify a job’s dependency list when the job is rejected.
- Modify a job’s list of environment variables before it gets to the execution host(s).

6.7.6 Controlling Interactive Jobs

- Control interactive job submission; for example, enable or disable interactive jobs at the server or queue level.

6.7.7 Communicating Information to Users

- For a job that is rejected because of a license shortage, set the job’s comment to inform about the shortage
- Report useful error messages back to the user, e.g., "You do not have sufficient walltime left to run your job for 1:00:00. Your walltime balance is 00:30:00."

6.7.8 Converting Requests to Usable Format

- Convert from ALPS-specific resource request strings into PBS-specific job requirements.
- Automatically translate old syntax to new syntax.
6.7.9 Helping Schedule Jobs

- Increase the priority of an array job once the first subjob runs, by modifying the value of a job resource used in the job sorting formula.
- Change scheduling according to user and job:
  - Set initial user-dependent coefficients for the scheduling formula. For example, set values of custom resources based on job attributes and user.
  - Set whether or not the job is rerunnable, based on user.
  - Calculate CPH ($CPH = \text{total ncpus} \times \text{walltime in hours}$) and set a custom CPH job resource to the value.
- Set initial priorities for jobs.

6.7.10 Managing User Activity

- Reject jobs from blacklisted users.
- Prevent users from using `qalter` to change their jobs in any way, allowing only administrators to `qalter` jobs.
- Prevent users from bypassing controls: disallow a job being submitted to queueA in a held state and then being moved to queueB where the job would not have passed hook checks for queueB initially. For example, if a `queuejob` hook disallows interactive jobs for queueB, the administrator also needs to ensure that an interactive job is not initially submitted to queueA and later moved to queueB.
- Prevent users from overriding `node_group_key` with `qsub -lplace = group = X`, or with `qalter`.
- Restrict the ability to submit a reservation to PBS administrators only.

6.7.11 Enabling Accounting and Validation

- Make sure correct project designation is used: if no project or account string is found, look up username in database to find appropriate project to use and add it as project or account string before submission.
- Submit job to correct queue based on project: check for project number and submit job to queues based on project type, e.g. project number 1234 jobs get submitted into “challenge” queue; similarly for “standard” queue, etc.
- Validate project before the job executes; if validation fails, do not start job, and print error message. Validation can be based on project name, and things like requested resources, for example, CPU hours.
6.7.12 Enforcing Security

- Reject jobs with invalid Kerberos tickets.

6.7.13 Provisioning Vnodes

- Provision a vnode with a new AOE. See Chapter 7, "Provisioning", on page 597.

6.7.14 Managing Job Output

- Manage where output goes by modifying a job’s output path with the job’s ID.

6.7.15 Allocation Management

- You can use a job submission (queuejob) hook to check whether an entity has enough resources allocated to accept the job.
- You can use a hook that runs just before the job is sent to the execution host (runjob) to perform allocation management tasks such as deducting requested amounts of resources from an entity’s allocation.
- You can use a hook that runs after a job finishes (execjob_epilogue) to perform final allocation management tasks such as allocation reconciliation.

6.7.16 Helping Manage Job Execution

Hooks that run at execution hosts can do the following:

- Modify job environment variables
- Check vnode health
- Report I/O wait time
- Report memory usage integral (MB*time used)
- Report energy usage to run a given job, if you have power sensors on vnodes
- Report actual usage of accelerator hardware (FPGAs, GPUs, etc)
- Interrogate HW performance counters so that you can flag codes that are not running efficiently (e.g. FLOPS < 5% of peak FLOPS)
- Record how much disk space a job has accumulated in PBS_JOBDIR
- Record power usage, energy usage, and disk space usage
6.8 When, as Whom, and Where Hooks Run

6.8.1 Overview of When Hooks Run

Each type of event has a corresponding type of hook. The following are the events where you can run hooks, with the hook type:

- Hooks that run before a job is received by an execution host (pre-execution event hooks):
  - queuejob: Queueing a job
  - modifyjob: Modifying a job (can also run after job is received by execution host)
  - movejob: Moving a job
  - runjob: Just before a job is sent to an execution host
- Hooks that run after a job is received by an execution host (execution event hooks):
  - execjob_begin: When a job is received by an execution host, after stagein
  - execjob_prologue: Just before executing a job
  - execjob_epilogue: Just after executing a job
  - execjob_preterm: Just before killing a job
  - execjob_end: Just after killing a job
- Hooks that are not directly related to a specific job (non-job event hooks):
  - resvsub: Submitting a PBS reservation
  - provision: Provisioning a vnode
  - exechost_periodic: Periodically on all execution hosts
Figure 6-3: Simplified view of hook trigger timing
Each time an event triggers a hook, the hook runs for that instance of the event. If you have written a hook that runs at job submission, this hook will run for each job that is submitted to this server. Each MoM runs one copy of each of her execution hooks per job. Execution hooks run one per job at the MoM, not one per vnode. For a job that runs on four vnodes of a multi-vnoded machine where all the vnodes are managed by one MoM, where you have written one execution hook, only one instance of the hook runs for that job.

Each time a job goes through a triggering event, PBS runs any relevant hooks. This means that if you run a job, that triggers a runjob hook. If the job is killed and requeued and runs again, the runjob hook runs again.

When you are using peer scheduling, and a job is pulled from one complex to another, the pulling complex applies its hooks as if the job had been submitted locally, and the furnishing complex applies its movejob hooks. Figure 6-4 shows an example of the hooks that are triggered when a job is moved from a complex containing a movejob hook to a complex containing a queuejob hook.

Figure 6-4: Hooks that run when job is moved
6.8.2 Execution Event Hook Triggers in Lifecycle of Job

The following list shows where execution event hooks are triggered in the lifecycle of a normal, successful job.

1. Licenses are obtained
2. Any required job-specific staging and execution directories are created
3. PBS_JOBDIR and job’s jobdir attribute are set to pathname of staging and execution directory
4. Files are staged in
5. MoM receives the job
6. Any execjob_begin hooks are triggered
7. Server writes accounting log “S” record
8. MoM tells sister MoMs they will run part of job
9. MoM creates work directory, temporary directory
10. MoM sets TMPDIR, JOBDIR, and other environment variables in job’s environment
11. MoM performs hardware-dependent setup:
    - The job’s cpusets are created
    - ALPS reservations are created
12. If any execjob_prologue hooks exist, they are triggered. If not, and there is a prologue, the prologue is executed
13. The job script is executed
14. The job script finishes
15. If any execjob_epilogue hooks exist, they are triggered. If not, and there is an epilogue, the epilogue is executed
16. The obit is sent to the server
17. Server writes accounting log “E” record
18. Any specified file staging out takes place, including stdout and stderr
19. Files staged in or out are deleted
20. Any job-specific staging and execution directories are removed
21. Any execjob_end hooks are triggered
Chapter 6

Hooks

22. The job’s cpusets are destroyed
23. Job files are deleted
24. Application licenses are returned to pool

6.8.3 Account Under Which Hooks Run

A hook runs as the Administrator or as a user, depending on the value of the hook’s user attribute. If this is set to pbsadmin, the hook runs as the Administrator. If this is set to pbsuser, the hook runs as the job owner.

6.8.4 Where Hooks Run

Hooks run on the primary or secondary server’s host or on the execution host(s).

6.9 What Hooks Do

Hooks accept (allow) or reject (prevent) actions, modify input parameters and job attributes, and change internal or external values. Hooks work with both the primary and secondary servers during failover. Hooks can only be created, run, or modified by the Administrator, and only on the hosts on which the servers run.

6.9.1 Accepting or Rejecting Actions

Each action can have zero or more hooks. Each hook must either accept or reject its action. All of an action’s hooks are run when that action is to be performed. For PBS to perform an action, all hooks enabled for that action must accept the action. If any hook rejects the action, the action is not performed by PBS. If a hook script doesn’t call accept() or reject(), and it doesn’t encounter an exception, PBS behaves as if the hook accepts the action. An action is always accepted, unless:

- pbs.event().reject() is called
- An unhandled exception is encountered
- The hook alarm has been triggered due to hook timeout being reached

When PBS executes the hooks for an action, it stops processing hooks at the first hook that rejects the action.
6.9.1.1 Examples of Accepting and Rejecting Actions

Example 6-2: Accepting an action: In this example, userA submits a job to queue Queue1, and the job submission action has two hooks: hook1 disallows jobs submitted by UserB, and hook2 disallows jobs being submitted directly to Queue2. Both hook1 and hook2 accept userA's job submission to Queue1, so the submission goes ahead.

Example 6-3: Rejecting an action: In this example, userA uses the qmove command to try to move jobA from Queue1 to Queue2. The job move action has two hooks: hook3 disallows jobs being moved into Queue2, and hook4 disallows userB moving jobs out of Queue1. In this example, hook3 rejects the action, so the move operation is disallowed, even though hook4 would have accepted the action.

6.9.2 Modifying Job Submission (qsub)

- When a job is submitted via qsub, queuejob hooks can modify the following things explicitly specified in the job submission:
  - Job attributes that can be set via qsub
  - Requested resources
- When a job is submitted via qsub, queuejob hooks can add resource requests to those specified in the job submission
- The input job attributes on which queuejob hooks operate are those that exist after all qsub processing is completed. These input attributes include:
  - Command line arguments
  - Script directives
  - Server default_qsub_arguments
- When a queuejob hook runs at job submission, the hook can affect only that job.
- For queuejob hooks, the input job attributes do not include:
  - Server or queue resources_default or default_chunk.
  - Conversions from old syntax (-lnodes or -lncpus) to new select and place syntax

6.9.3 Modifying Job Change (qalter)

- When a job is changed via qalter, modifyjob hooks can modify the arguments passed to qalter
- When a modifyjob hook runs, it can change the attributes of the job that can be changed via qalter
6.9.4 Modifying Job Move (qmove)

- When a job is moved via qmove, movejob hooks can modify the arguments passed to qmove.
- When a movejob hook runs, it can change the job’s destination queue to any queue on the default server.

6.9.5 Modifying Reservation Creation (pbs_rsub)

- When an advance or standing reservation is created via pbs_rsub, resvsub hooks can modify the reservation’s attributes that can be set via pbs_rsub.
- When an advance or standing reservation is created, resvsub hooks can specify additional attributes that can be specified via pbs_rsub.
- The input reservation attributes on which resvsub hooks operate are those that exist after all pbs_rsub processing of command line arguments is completed.
- For resvsub hooks, the input job attributes do not include:
  - server or queue resources_default or default_chunk.
  - conversions from old syntax (-lnodes & -lncpus) to new select and place syntax.

6.9.6 Changes Before Job is Sent to MoM (qrun)

When the scheduler runs a job or the administrator runs a job using the qrun command, any runjob hooks are executed.

- When a runjob hook accepts a job, it can do the following:
  - Set the job’s Output_Path attribute
  - Set the job’s Error_Path attribute
  - Set some of the job’s Resource_List members; see section 6.13.5.4.i, “Restrictions.”
Hooks

Chapter 6

6.9.7 Changes When Job is Received by MoM

When MoM receives a job, an execjob_begin hook can:

- Modify the job’s Execution_Time, Hold_Types, Variable_List, and resources_used attributes
- Flag the job to be rerun
- Kill the job
- Set attributes and resources on the vnode(s) managed by the MoM where this job executes

6.9.8 Changes Before Job is Executed

Just before a job is executed, an execjob_prologue hook can:

- Modify the job’s Execution_Time, Hold_Types, and resources_used attributes
- Flag the job to be rerun
- Kill the job
- Set attributes and resources on the vnode(s) managed by the MoM where this job executes

- Set any members of the job’s Variable_List attribute
- When a runjob hook rejects a job, it can do the following:
  - Set the job’s depend attribute
  - Set any members of the job’s Variable_List attribute
  - Place a hold on the job
  - Release a hold on the job
  - Set the job’s project attribute
  - Change the time the job is allowed to begin execution
  - Set any of the job’s Resource_List attribute members except nodect
  - Change the state of a vnode where the job would have run
6.9.9 Changes After Job is Executed

Just after a job is executed, an execjob_epilogue hook can:

- Modify the job’s Execution_Time, Hold_Types, and resources_used attributes
- Flag the job to be rerun
- Kill the job
- Set attributes and resources on the vnode(s) managed by the MoM where this job executes
- Use the job’s exit status

6.9.10 Changes Before Job is Killed

Just before a job is killed, an execjob_preterm hook can:

- Modify the job’s Execution_Time, Hold_Types, and resources_used attributes
- Flag the job to be rerun
- Kill the job
- Set attributes and resources on the vnode(s) managed by the MoM where this job executes
- Cause the job to keep running

6.9.11 Changes After Job Finishes or is Killed

Just after a job finishes execution or is killed, an execjob_end hook can:

- Set attributes and resources on the vnode(s) managed by the MoM where this job executes
- Use the job’s exit status

6.9.12 Periodic Events at Execution Hosts

- An exechost_periodic hook can set attributes and resources for any vnode managed by the MoM on the host where the hook runs. This means that an instance of a hook can affect more than one vnode only when the hook is running on a multi-vnode host.
6.9.13 Failover

When the secondary server takes over for the primary server after the primary's host has gone down or becomes inaccessible, any hooks registered with the primary server continue to function under the secondary server.

Likewise, if the primary server comes back up and takes over, hooks registered while the secondary server had control continue to function.

The secondary server uses the same filesystem as the primary server. If the administrator creates a new hook while the secondary server has control, that hook will persist once the primary server takes over. Any hooks created, whether the primary or the secondary server is running, are stored in the same place, and are accessible by both servers.

6.10 Creating Hooks

6.10.1 Introduction to Creating and Deleting Hooks

Please read section 6.5, “Updating Hooks and Custom Resources”, on page 419.

Hooks can only be created, run, or modified by the Administrator, and only on the host on which the server runs.

You create hooks using the `qmgr` command to create, delete, import, or export the hook. The `qmgr` command operates on the `hook` object.

Format of `qmgr` hooks directive:

```
command hook [hook_name] [attr OP value[,attr OP value,...]]
command is create, delete, set, unset, list, print, import, export
import  loads the contents of a hook from an input file.
export  dumps the hook contents to a file.
```
Hooks

6.10.1.1 Overview of Creating a Hook

The following is an overview of the steps to create a hook. Each step is described in the following sections.

1. Use the `create hook qmgr` command to create an empty hook with the name you specify.
2. Set the hook’s trigger event.
3. Import the contents of a hook script into the hook.
4. Set the hook’s order of execution, if there is another hook for the same event.
5. Optionally, set the hook’s timeout.
6. Make sure that the `$reject_root_scripts` MoM configuration parameter is set to `False` on all execution hosts where you want hooks to run. The default for this parameter is `False`.

You do not need to restart the MoM.

6.10.1.1.i Hook Name Requirements

Each hook has a unique name. This name must be alphanumeric, and start with an alphabetic character. The name must not begin with "$PBS".

The name of a hook can be a legal PBS object name, such as the name of a queue. Hook names are case-sensitive.

6.10.1.1.ii Example of Creating a Hook

Create the hook:

```
Qmgr: create hook hook1
```

Import the hook script named `hook1_script.py` into the hook:

```
Qmgr: import hook hook1 application/x-python default /hooks/hook1_script.py
```

Make hook1 a `queuejob` hook:

```
Qmgr: set hook hook1 event = queuejob
```

Make this the second `queuejob` hook:

```
Qmgr: set hook hook1 order = 2
```

Set the hook to time out after 60 seconds:

```
Qmgr: set hook hook1 alarm = 60
```
Look at the $reject_root_scripts MoM configuration parameter where you want the hook to run, and make sure it is set to False.

6.10.2 Creating Empty Hooks

To create a hook, use the create hook qmgr command to create an empty hook with the name you specify:

The create hook qmgr command creates an empty hook.

Format for creating a hook:

Qmgr: create hook <hook name>

6.10.2.1 Example of Creating an Empty Hook

To create the hook named “hook1”, specify a filename, for example “/hooks/hook1.py”, that is locally accessible to qmgr and the PBS Server:

Qmgr: create hook hook1

6.10.3 Deleting Hooks

To delete a hook, you use the delete hook qmgr command.

Format for deleting a hook:

Qmgr: delete hook <hook name>

6.10.3.1 Example of Deleting a Hook

To delete hook hook1:

Qmgr: delete hook hook1

6.10.4 Setting Hook Trigger Events

To set the events that will cause a hook to be triggered, use the set hook <hook name> event qmgr command. You can add triggering events to a hook.

To set one event:

Qmgr: set hook <hook name> event = <event name>
Designate triggers for a hook by setting `<event name>` to one of the following events:

**Table 6-1: Hook Trigger Events**

<table>
<thead>
<tr>
<th>Action (Event)</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accepting job into queue</td>
<td>queuejob</td>
</tr>
<tr>
<td>Modifying job</td>
<td>modifyjob</td>
</tr>
<tr>
<td>Moving job</td>
<td>movejob</td>
</tr>
<tr>
<td>Before a job is sent to an execution host</td>
<td>runjob</td>
</tr>
<tr>
<td>When a job is received by an execution host after stagein</td>
<td>execjob_begin</td>
</tr>
<tr>
<td>Just before executing a job</td>
<td>execjob_prologue</td>
</tr>
<tr>
<td>Just after executing a job</td>
<td>execjob_epilogue</td>
</tr>
<tr>
<td>Just before killing a job</td>
<td>execjob_preterm</td>
</tr>
<tr>
<td>Just after killing a job</td>
<td>execjob_end</td>
</tr>
<tr>
<td>Periodically on all execution hosts</td>
<td>exechost_periodic</td>
</tr>
<tr>
<td>Provisioning a vnode</td>
<td>provision</td>
</tr>
<tr>
<td>Submitting reservation</td>
<td>resvsub</td>
</tr>
</tbody>
</table>

To add an event:

```
Qmgr: set hook <hook name> event += <event name>
```

**6.10.4.1 Example of Setting Hook Trigger Events**

To set the events that will cause hook “UserFilter” to be triggered:

```
Qmgr: set hook UserFilter event = queuejob
Qmgr: set hook UserFilter event += modifyjob
```

Set two events at once:

```
Qmgr: set hook UserFilter event = “queuejob, modifyjob”
```

You must enclose the value in double quotes if it contains a comma.
6.10.5 Importing Hooks

To import a hook, you import the contents of a hook script into the hook. You must specify a filename that is locally accessible to qmgr and the PBS Server.

Format for importing a hook:

```
qmgr: import hook <hook_name> <content-type> <content-encoding>
    {<input_file>|stdin}
```

This uses the contents of `<input_file>` or `stdin` as the contents of hook `<hook_name>`.

- The `<input_file>` or `stdin` data must have a format `<content-type>` and must be encoded with `<content-encoding>`.
- The only `<content-type>` currently supported is "application/x-python".
- The allowed values for `<content-encoding>` are "default" (7 bit) and "base64".
- If the source of input is `stdin` and `<content-encoding>` is "default", then `qmgr` expects the input data to be terminated by EOF.
- If the source of input is `stdin` and `<content-encoding>` is "base64", then `qmgr` expects input data to be terminated by a blank line.
- `<input_file>` must be locally accessible to both `qmgr` and the requested batch server.
- A relative path in `<input_file>` is relative to the directory where `qmgr` was executed.
- If a hook already has a content script, then that is overwritten by this import call.
- If the name of `<input_file>` contains spaces, as are used in Windows filenames, then `<input_file>` must be quoted.
- There is no restriction on the size of the hook script.
6.10.5.1 Examples of Importing Hooks

Example 6-4: Given a Python script in ASCII text file "hello.py", this makes its contents into the script contents of hook1:

```
# cat hello.py
import pbs
pbs.event().job.comment="Hello, world"
# qmgr -c 'import hook hook1 application/x-python default hello.py'
```

Example 6-5: Given a base64-encoded file "hello.py.b64", qmgr unencodes the file's contents, and then makes this the script contents of hook1:

```
# cat hello.py.b64
bGVzc2l0ZSB2YXJ0aW9uIHRvYW4uc215
# qmgr -c 'import hook hook1 application/x-python base64 hello.py.b64'
```

Example 6-6: Read stdin for text containing data until EOF, and make this into the script contents of hook1:

```
# qmgr -c 'import hook hook1 application/x-python default -'
import pbs
pbs.event().job.comment="Hello from stdin"
Ctrl-D (UNIX/Linux)
Ctrl-Z (Windows)
```

Example 6-7: Read stdin for a base64-encoded string of data terminated by a blank line. PBS unencodes the data and makes this the script contents of hook1.

```
# qmgr -c 'import hook hook1 application/x-python base64 -'
bGVzc2l0ZSB2YXJ0aW9uIHRvYW4uc215
Ctrl-D (UNIX/Linux)
Ctrl-Z (Windows)
```

6.10.6 Exporting Hooks

Format for exporting a hook:

```
Qmgr: export hook <hook_name> <content-type> <content-encoding> [<output_file>]
```

This dumps the script contents of hook <hook_name> into <output_file>, or stdout if <output_file> is not specified.

- The resulting <output_file> or stdout data is of <content-type> and <content-encod-
Hooks

6.10.6.1 Examples of Exporting Hooks

Example 6-8: Dumps hook1's script contents directly into a file "hello.py.out":

```bash
# qmgr -c 'export hook hook1 application/x-python default hello.py'
# cat hello.py
import pbs
pbs.event().job.comment="Hello, world"
```

Example 6-9: To dump the script contents of a hook 'hook1' into a file in "\My Hooks\hook1.py":

```bash
Qmgr: export hook hook1 application/x-python default "\My Hooks\hook1.py"
```

Example 6-10: Dump hook1's script contents base64-encoded into a file called "hello.py.b64":

```bash
# qmgr -c "export hook hook1 application/x-python base64 hello.py.b64"
# cat hello.py.b64
cHJpbnQgImhlbGxvLCB3b3JsZCIK
```

Example 6-11: Dump hook1's script contents directly to stdout:

```bash
# qmgr -c "export hook hook1 application/x-python default"
import pbs
pbs.event().job.comment="Hello, world"
```

Example 6-12: Dump hook1's script contents base64-encoded into stdout:

```bash
# qmgr -c "export hook hook1 application/x-python base64"
cHJpbnQgImhlbGxvLCB3b3JsZCIK
```
6.11 Configuring Hooks

6.11.1 Hook Attributes

You configure a hook using the `qmgr` command to set or unset its attributes. An unset hook attribute takes the default value for that attribute.

Hook attributes can be viewed via `qmgr`:

```
Qmgr: list hook <hook name>
```

### 6.11.1.1 Setting and Unsetting Hook Attributes

To set a hook attribute:

```
Qmgr: set hook <hook name> <attribute> = <value>
```

To unset a hook attribute:

```
Qmgr: unset hook <hook name> <attribute>
```

For example, to unset hook1’s `alarm` attribute, causing its value to revert to its default value:

```
Qmgr: unset hook hook1 alarm
```

This causes hook1’s `alarm` to revert to the default of 30 seconds.

### 6.11.1.2 List of Hook Attributes

Hook attributes are listed in the following table:

<table>
<thead>
<tr>
<th>Attribute Name and Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>alarm=&lt;n&gt;</code></td>
<td><code>&lt;n&gt;</code> is the number of seconds to wait before an executing hook script times out. Valid values are &gt; 0. Default value: 30</td>
</tr>
</tbody>
</table>
Hooks

Chapter 6

Table 6-2: Hook Attributes

<table>
<thead>
<tr>
<th>Attribute Name and Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>enabled=&lt;Boolean&gt;</code></td>
<td>Determines whether or not a hook is run when its triggering event occurs. If a hook's <code>enabled</code> attribute is <code>True</code>, the hook is run. Default value: <code>True</code></td>
</tr>
</tbody>
</table>
| `event=<event_string_array>` | List of events that trigger the hook. Can be operated on with “=”, “+=”, or “-=” operators. Valid hook types are:  
  - `queuejob`: submit a job  
  - `modifyjob`: alter a job  
  - `movejob`: move a job  
  - `runjob`: before sending job to execution host  
  - `execjob_begin`: when execution host receives job  
  - `execjob_prologue`: before running job  
  - `execjob_epilogue`: after job finishes  
  - `execjob_preterm`: before killing job  
  - `execjob_end`: after cleaning up job  
  - `exechost_periodic`: periodically on all execution hosts  
  - `resvsub`: create a reservation  
  - `provision`: provision a vnode  
  The `provision` event cannot be combined with any other events. See section 6.13.8.2, “Event Type Objects”, on page 486. Default value: `=none`, meaning the hook will not be triggered. |
6.11.2 Setting Hook Trigger Events

See section 6.10.4, “Setting Hook Trigger Events”, on page 441.
6.11.3 Enabling and Disabling Hooks

A hook is either enabled, and will run when its action happens, or is disabled, and will not run. Hooks are enabled by default.

Format to enable a hook:

```
Qmgr: set hook <hook name> enabled=True
```

Format to disable a hook:

```
Qmgr: set hook <hook name> enabled=False
```

6.11.3.1 Example of Enabling and Disabling Hooks

To enable hook1:

```
Qmgr: set hook hook1 enabled=True
```

To disable hook1:

```
Qmgr: set hook hook1 enabled=False
```

6.11.4 Setting the Relative Order of Hook Execution

When there are multiple hooks for one action, you may wish to specify the order in which the hooks are run. The order in which the hooks for an action are run is determined by each hook’s order attribute. Hooks with a lower value for order will run before hooks with a higher value. To set the relative order in which the hooks for an action will be run, set each hook’s order attribute.

Format:

```
Qmgr: set hook <hook name> order=<ordering>
```

<ordering> is an integer. Hooks with lower values for <ordering> run before those with higher values; a hook with order=1 runs before a hook with order=2.

Valid values for hook ordering are between 1 and 1000.
Chapter 6  

Hooks

The order in which hooks for unrelated actions execute is undefined. For example, there are two `queuejob` hooks, Hook1 and Hook2, and userA submits jobA and userB submits jobB. While Hook1 always runs before Hook2 for the same job, the order of execution is undefined for different jobs. So the order could be:

- Hook1 (jobB)
- Hook1 (jobA)
- Hook2 (jobA)
- Hook2 (jobB)

6.11.4.1 Example of Setting Relative Order of Hook Execution

To set hookA to run first and hookB to run second:

- `Qmgr: set hook hookA order=2`
- `Qmgr: set hook hookB order=5`

6.11.4.2 Caveats for Setting Relative Order of Hooks

The `order` attribute is ignored for `exechost_periodic` hooks.

6.11.5 Setting Hook Timeout

You may wish to specify how long PBS should wait for a hook to run. Execution for each hook times out after the number of seconds specified in the hook’s `alarm` attribute. If the hook does not run in the specified time, PBS aborts the hook and rejects the hook’s action.

Format:

- `Qmgr: set hook <hook name> alarm=<timeout>`

`<timeout>` is the number of seconds PBS will allow the hook to run.

When a hook timeout is triggered, the hook script gets a Python `KeyboardInterrupt` from the PBS server. The server logs show:

```
06/17/2008 17:57:16;0001;Server@host2;Svr;Server@host2;PBS server internal error (15011) in Python script received a KeyboardInterrupt, <type 'exceptions.KeyboardInterrupt'>
```
6.11.5.1 Example of Setting Hook Timeout

To set the number of seconds that PBS will wait for hook hook1 to execute before aborting the hook and reject the action:

```
Qmgr: set hook hook1 alarm=20
```

6.11.6 Setting Hook Frequency

You can specify the frequency with which a periodic hook runs. You can do this only for hooks whose event type is `exechost_periodic`.

Format:

```
Qmgr: set hook <hook name> freq=<frequency>
```

`<frequency>` is the number of seconds elapsed between calls to this hook.

6.11.6.1 Example of Setting Hook Frequency

To set the number of seconds between calls to an `exechost_periodic` hook:

```
Qmgr: set hook hook1 freq=200
```

6.11.7 Setting Hook User Account

You can specify the account under which a hook runs.

Format:

```
Qmgr: set hook <hook name> user=<pbsadmin | pbsuser>
```

`pbsadmin` specifies that the hook runs as root or as administrator.

`pbsuser` specifies that the hook runs as the job owner.

You can specify that a hook runs as the job owner only for `execjob_prologue`, `execjob_epilogue`, and `execjob_preterm` hooks.

If you do not set the account, it defaults to `pbsadmin`.

6.11.7.1 Example of Setting Hook User Account

To set the account under which a hook runs:

```
Qmgr: set hook hook1 user=pbsuser
```
6.12 Viewing Hook Information

6.12.1 Listing Hooks

To list one hook and its attributes on the current server:

Qmgr: list hook <hook name>

To list all hooks and their attributes on the current server:

Qmgr: list hook

6.12.2 Viewing Hook Contents

To view the contents of a hook, export the hook’s contents:

Qmgr: export hook <hook_name> <content-type> <content-encoding> [output_file]

6.12.3 Printing Hook Creation Commands

To view the commands to create one hook:

Qmgr: print hook <hook name>

To view the commands to create all the hooks on the default server:

Qmgr: print hook

or

qmgr -c “print hook”
For example, to see the commands used to create hook1 and hook2:

```bash
# qmgr -c "print hook"
create hook hook1
import hook hook1 application/x-python base64 -
   cHJpbnQgImhlbGxvLCB3b3JsZCIIK<blank line>
set hook hook1 event=movejob
set hook hook1 alarm=10
set hook hook1 order=5
create hook hook2
import hook hook2 application/x-python base64 -
   aservaJLSDFSESF<newline>
set hook hook2 event=queuejob
set hook hook2 alarm=15
set hook hook2 order=60
```

6.12.4 Re-creating Hooks

To re-create a hook, you feed `qmgr` hook descriptions back into `qmgr`. These hook descriptions are the same information that `qmgr` prints out. To print out the statements needed to recreate a hook, use the `print hook` or `print hook <hook name>` `qmgr` commands.

For example, to save information for hook1 and hook2:

```bash
# qmgr -c "print hook" > hookInfo
```

To re-create hook1 and hook2:

```bash
# qmgr < hookInfo
```
6.13 Interface to Hooks

6.13.1 The pbs Module

The pbs module provides an interface to PBS and the hook environment. The interface is made up of Python objects, which have members and methods. You can operate on these objects using Python code. In order to use the pbs module, you must begin your Python code by importing the module. For example, in a script that modifies a job:

```python
import pbs
pbs.event().job.comment="Modified this job"
```

For the contents of the pbs module, see section 6.16, “Python Modules and PBS”, on page 566.

6.13.2 What Hooks Can Access and Do

- Hooks can allow or disallow the action for which the hook is executing
- Pre-execution and execution hooks can modify certain job attributes and resources. See section 6.13.5, “Using Attributes and Resources in Hooks”, on page 456 for a complete list of which attributes and resources can be modified by each hook.
- Hooks can change the following for certain job operations:
  - Job attributes and resources specified for job submission
  - Arguments to qalter or qmove
  - Before a job is sent to the execution host, runjob hooks can alter the job’s execution time, project, output and error paths, hold types, dependencies, environment variables, and some members of the job’s Resource_List, and change the state of any vnode listed in the job’s exec_vnode attribute
  - Hooks can read the user attempting to execute the action; the Server, Scheduler, and MoMs are treated as users
  - resvsub hooks can read and modify the reservation attributes settable via pbs_rsub, during reservation creation
  - Hooks can read environment variables from the user's session
  - Hooks can read all Server attributes for the local server
  - Hooks can read all queue attributes for any queue on the local server
  - Hooks can specify a message to be printed by the daemon where the event takes place
  - Pre-execution hooks can read vnode attributes and global resources; the only vnode
attribute that can be set by pre-execution event hooks is the `state` vnode attribute

- Execution hooks can read and set any settable vnode attributes and resources

### 6.13.3 What Hooks Cannot Access or Do

- Hooks cannot read or modify anything not presented in the PBS hook interface
- Hooks cannot modify the server or any queues
- Pre-execution event hooks cannot read or set vnode attributes or resources, except that the `runjob` hook can set the `state` attribute for any vnode to be used by the job
- Hooks do not have access to other servers besides the default server:
  - Hooks cannot change the destination server to a non-default server
  - Hooks can allow a job submission or a `qmove` to a non-default server, and can change the destination server from a remote server to the default server
- Hooks cannot directly print to `stdout` or `stderr` or read from `stdin`
- `movejob` hooks do not run on `pbs_rsub -Wqmove=<job ID>`

### 6.13.4 What Hooks Should Not Do

- Hooks should not edit configuration files directly, meaning hooks should not edit the following:
  - `PBS_HOME/sched_priv/sched_config`
  - `PBS_HOME/sched_priv/fairshare`
  - `PBS_HOME/sched_priv/dedicated`
  - `PBS_HOME/sched_priv/holidays`
  - `/etcpbs.conf`
  - `PBS_HOME/server_priv/resourcedef`
  - `PBS_HOME/mom_priv/config`
- Hooks should not execute PBS commands
6.13.5 Using Attributes and Resources in Hooks

6.13.5.1 How Object is Retrieved Affects Access

The way you retrieve an object determines how much access you have to that object. You can retrieve an object either through the event, via `pbs.event().job`, or through the server, via `pbs.server().job()`.

If you retrieve a job through an event, the event gives you the job itself, represented as an object. You can see and alter some job attributes for an event-retrieved job object. However, if you retrieve a job through the server, the server gives you an instantiated job object that contains a copy of the job. You can see all of the attributes for a server-retrieved job object, except in a `queuejob` hook. In a `queuejob` hook, the event gives you the job as it exists before the server sees it, but the server cannot retrieve it, because the job has not yet made it to the server. You cannot set any job attributes for a server-retrieved job object, and trying to operate on a server-retrieved copy of the job causes an exception. We show which hooks can see and set each job attribute in Table 6-3, “Job Attributes Readable & Settable via Events,” on page 458.

Vnode objects behave like job objects. If you retrieve a vnode object through an event, via `pbs.event().vnode_list[]`, you can see some of the vnode’s attributes, and set any vnode attribute that you could set via `qmgr`. However, if you retrieve a vnode object through the server, via `pbs.server().vnode()`, you have a copy of the vnode, and you can see all of the vnode’s attributes, but you cannot set any of them. We show which hooks can see and set each vnode attribute in Table 6-4, “Vnode Attributes Readable & Settable via Events,” on page 463.

6.13.5.2 How to Set Attributes and Resources in Hooks

The way you set a job, vnode, or reservation attribute depends on the attribute’s type:

- If the attribute is a string (str), an integer (int), a Boolean (bool), a long (long), or a floating point (float), you can set it directly, for example:
  ```
  jobA = pbs.event().job
  jobA.Account_Name = "AccountA"
  jobA.Priority = 100
  ```

- However, if the attribute is any other type, you must first use the associated creation method to instantiate an object of the correct type with the desired value, then assign the object to the job. For example:
  ```
  jobB = pbs.event().job
  jobB.Hold_Types = pbs.hold_types("uo")
  ```

456 PBS Professional 12.2 Administrator’s Guide
6.13.5.2.i Python Types not Requiring Explicit Creation Method

The following Python types do not require you to use an explicit creation method:

- str
- int
- bool
- long
- float

6.13.5.3 Using Attributes in Hooks

All hooks can read, but not set, all job, vnode, server, queue, and reservation attributes via `pbs.server().job()`, `pbs.server().vnode()`, `pbs.server().queue()`, etc. We list which job attributes can be read or set when the job is retrieved through an event in Table 6-3, “Job Attributes Readable & Settable via Events,” on page 458. We list which vnode attributes can be read or set when the vnode is retrieved through an event in Table 6-4, “Vnode Attributes Readable & Settable via Events,” on page 463. We list which reservation attributes can be read or set when the reservation is retrieved through an event in Table 6-5, “Reservation Attributes Readable & Settable via Events,” on page 465. To summarize tables 6-3, 6-4, and 6-5:

- The only job attribute that a movejob event hook can set is the job’s destination queue. The hook shows the job’s current queue in the `pbs.event().src_queue` object member.

- A modifyjob event hook shows the original job with all its attributes in `pbs.event().job_o`. This hook can set any settable job attribute before the job runs. While the job is running, the only job attributes that the hook can set are those that can be changed by an unprivileged user via the `qalter` command. See “qalter” on page 127 of the PBS Professional Reference Guide and “Job Attributes” on page 375 of the PBS Professional Reference Guide.

- The only job attributes that a runjob event hook can set are the `Hold_Types`, `project`, `Error_Path`, `Output_Path`, `Resource_List`, `Variable_List`, and `Execution_Time` job attributes.

- The only vnode attribute that a runjob event hook can set is the `state` vnode attribute. The only pre-execution event hook that can change this attribute is a runjob hook.

- A queuejob event hook can set certain job attributes.

- A resvsub event hook can set any reservation attribute or resource that can be specified via `pbs_rsub`. See Table 6-5, “Reservation Attributes Readable & Settable via Events,” on page 465.

- An execjob_begin, execjob_prologue, execjob_preterm, or execjob_epilogue event
hook can set the job’s Execution_Time, Hold_Types, and resources_used attributes. It can also modify the attributes and available resources of the vnodes managed by the local MoM.

- An execjob_end event hook can modify the attributes and available resources of the vnodes managed by the local MoM.
- A queuejob, modifyjob, or execjob_begin event hook can set the job’s Variable_List attribute. Other hooks may appear to be able to set this attribute, but the changes will not affect the job.
- An execjob_epilogue or execjob_end event hook can capture and use the job’s exit status.
- An exechost_periodic event hook can set vnode attributes for vnodes managed by the local MoM.

No hooks can see or set any scheduler attributes.

The job, vnode, or reservation object’s attributes appear to the hook as they would be after the event, not before it, for all hooks except runjob hooks.

### 6.13.5.3.i Using Job Attributes in Hooks

Job attributes can be read and set in pre-execution and execution hooks. Non-job-event hooks (provisioning, reservation, and periodic) can read job attributes, but do not have an associated job, so they cannot set job attributes. All hooks can read but not set all job attributes using pbs.server().job(). The following table lists the job attributes that can be read or set when the job is retrieved via an event:

**Table 6-3: Job Attributes Readable & Settable via Events**

<table>
<thead>
<tr>
<th>Job Attribute</th>
<th>queuejob</th>
<th>modifyjob (before run)</th>
<th>movejob</th>
<th>runjob (on reject)</th>
<th>runjob (on accept)</th>
<th>execjob_begin</th>
<th>execjob_prologue</th>
<th>execjob_end</th>
<th>execjob_epilogue</th>
<th>execjob_preterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account_Name</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>argument_list</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
</tbody>
</table>
Table 6-3: Job Attributes Readable & Settable via Events

<table>
<thead>
<tr>
<th>Job Attribute</th>
<th>queuejob</th>
<th>modifyjob (before run)</th>
<th>movejob</th>
<th>runjob (on reject)</th>
<th>runjob (on accept)</th>
<th>execjob_begin</th>
<th>execjob_prologue</th>
<th>execjob_end</th>
<th>execjob_epilogue</th>
<th>execjob_preterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>array_indices_submitted</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>block</td>
<td>---</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Checkpoint</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>comment</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>depend</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>s</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>eligible_time</td>
<td>---</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Error_Path</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Executable</td>
<td>r, s</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Execution_Time</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r, s</td>
<td>r</td>
<td>s</td>
<td>r, s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>exec_host</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>exec_vnode</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>group_list</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Hold_Types</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>s</td>
<td>r, s</td>
<td>r</td>
<td>s</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>interactive</td>
<td>r, s</td>
<td>r, s&lt;sup&gt;1&lt;/sup&gt;</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Job_Name</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Join_Path</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Keep_Files</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
</tbody>
</table>

<sup>1</sup> The symbols ‘r’ and ‘s’ denote read and set permissions, respectively.
### Chapter 6  
**Hooks**

<table>
<thead>
<tr>
<th>Job Attribute</th>
<th>queuejob</th>
<th>modifyjob (before run)</th>
<th>movejob</th>
<th>runjob (on reject)</th>
<th>runjob (on accept)</th>
<th>execjob_begin</th>
<th>execjob_prologue</th>
<th>execjob_end</th>
<th>execjob_epilogue</th>
<th>execjob_preterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mail_Points</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Mail_Users</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>no_stdio_sockets</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Output_Path</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Priority</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>project</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>pset</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Rerunable</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Resource_List (with restrictions; see 6.13.5.4.i)</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>run_count</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r, s</td>
<td>r, s</td>
</tr>
<tr>
<td>sandbox</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>sched_hint</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Shell_Path_List</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>stagein</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>stageout</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Submit_arguments</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>umask</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>User_List</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

#### Table 6-3: Job Attributes Readable & Settable via Events
### Table 6-3: Job Attributes Readable & Settable via Events

<table>
<thead>
<tr>
<th>Job Attribute</th>
<th>queuejob</th>
<th>modifyjob (before run)</th>
<th>movejob</th>
<th>runjob (on reject)</th>
<th>runjob (on accept)</th>
<th>execjob_begin</th>
<th>execjob_prologue</th>
<th>execjob_end</th>
<th>exitjob_epilogue</th>
<th>exitjob_preterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable_List</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
</tr>
<tr>
<td>Read-only attributes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>accounting_id</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>accru_type</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r, r</td>
<td>r, r</td>
<td>r, r</td>
<td>r, r</td>
<td>r</td>
</tr>
<tr>
<td>alt_id</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>array_id</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r, r</td>
<td>r, r</td>
<td>r, r</td>
<td>r, r</td>
<td>r</td>
</tr>
<tr>
<td>array_index</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r, r</td>
<td>r, r</td>
<td>r, r</td>
<td>r, r</td>
<td>r</td>
</tr>
<tr>
<td>array_indices_remaining</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>array_state_count</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>ctime</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>egroup</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r, r</td>
<td>r, r</td>
<td>r, r</td>
<td>r, r</td>
<td>r</td>
</tr>
<tr>
<td>estimated</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>etime</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>euser</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r, r</td>
<td>r, r</td>
<td>r, r</td>
<td>r, r</td>
<td>r</td>
</tr>
<tr>
<td>Exit_status</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>r, r</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>hostname</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r, r</td>
<td>r, r</td>
<td>r, r</td>
<td>r, r</td>
<td>r</td>
</tr>
<tr>
<td>jobdir</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Job_Owner</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
### Table 6-3: Job Attributes Readable & Settable via Events

<table>
<thead>
<tr>
<th>Job Attribute</th>
<th>queuejob</th>
<th>modifyjob (before run)</th>
<th>movejob</th>
<th>runjob (on reject)</th>
<th>runjob (on accept)</th>
<th>execjob_begin</th>
<th>execjob_prologue</th>
<th>execjob_end</th>
<th>execjob_epilogue</th>
<th>execjob_preterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_state</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>mtime</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>qtime</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>Queue</td>
<td>r, s r r r, s r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>queue_rank</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>queue_type</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>resources_used</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>run_version</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>schedselect</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>server</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>session_id</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>Stageout_status</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>stime</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>substate</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
<tr>
<td>sw_index</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
<td>--- r r r r r r r r r r</td>
</tr>
</tbody>
</table>

1. Not recommended; setting a value for the interactive job attribute in a modifyjob hook has no effect.
### 6.13.5.3.ii Using Vnode Attributes in Hooks

Vnode attributes can be read and set in execution hooks, periodic hooks, and the runjob hook. Other hooks can read vnode attributes by using `pbs.server().vnode()`, but do not have an associated vnode list, so they cannot set vnode attributes. The following table shows the vnode attributes that can be read or set when the vnode object is retrieved via an event:

**Table 6-4: Vnode Attributes Readable & Settable via Events**

<table>
<thead>
<tr>
<th>Vnode Attribute</th>
<th>queuejob</th>
<th>modifyjob (before run)</th>
<th>movejob</th>
<th>runjob</th>
<th>execjob_begin</th>
<th>execjob_prologue</th>
<th>execjob_end</th>
<th>execjob_epilogue</th>
<th>execjob_preterm</th>
<th>exechost_periodic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readable and settable attributes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>comment</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>current_aoe</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>hpcbp_enable</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>hpcbp_stage_protocol</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>hpcbp_user_name</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>hpcbp_webservice_address</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>max_group_run</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>max_running</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>max_user_run</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>Mom</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>no_multinode_jobs</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>ntype</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>pnames</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>Port</td>
<td></td>
<td></td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
</tbody>
</table>
### Table 6-4: Vnode Attributes Readable & Settable via Events

<table>
<thead>
<tr>
<th>Vnode Attribute</th>
<th>queuejob</th>
<th>modifyjob (before run)</th>
<th>movejob</th>
<th>runjob</th>
<th>execjob_begin</th>
<th>execjob_prologue</th>
<th>execjob_end</th>
<th>execjob_epilogue</th>
<th>execjob_preterm</th>
<th>exechost_periodic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>provision_enable</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>queue</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>s</td>
</tr>
<tr>
<td>resources_available</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>s</td>
</tr>
<tr>
<td>resv_enable</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>sharing</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>s</td>
</tr>
<tr>
<td>state</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>s</td>
</tr>
</tbody>
</table>

**Read-only attributes**

<table>
<thead>
<tr>
<th>Vnode Attribute</th>
<th>queuejob</th>
<th>modifyjob (before run)</th>
<th>movejob</th>
<th>runjob</th>
<th>execjob_begin</th>
<th>execjob_prologue</th>
<th>execjob_end</th>
<th>execjob_epilogue</th>
<th>execjob_preterm</th>
<th>exechost_periodic</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_multivnode_host</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>jobs</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>license</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>license_info</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>name</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>pbs_version</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>pcpus</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>resv</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>resources_assigned</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>topology_info</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
</tbody>
</table>
6.13.5.3.iii Using Reservation Attributes in Hooks

Reservation attributes can be read and set through an event only in resvsub hooks. No other hooks can read or set reservation attributes through an event. All hooks can read, but not set, all reservation attributes by retrieving the reservation object through the server, using pbs.server().resv(). The following table shows the reservation attributes that can be read or set when the reservation object is retrieved via an event, in a resvsub hook:

Table 6-5: Reservation Attributes Readable & Settable via Events

<table>
<thead>
<tr>
<th>Reservation Attribute</th>
<th>Readable, Settable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Readable and settable attributes</strong></td>
<td></td>
</tr>
<tr>
<td>Account_Name</td>
<td>r</td>
</tr>
<tr>
<td>Authorized_Groups</td>
<td>r, s</td>
</tr>
<tr>
<td>Authorized_Hosts</td>
<td>r, s</td>
</tr>
<tr>
<td>Authorized_Users</td>
<td>r, s</td>
</tr>
<tr>
<td>group_list</td>
<td>r, s</td>
</tr>
<tr>
<td>Mail_Points</td>
<td>r, s</td>
</tr>
<tr>
<td>Mail_Users</td>
<td>r, s</td>
</tr>
<tr>
<td>Priority</td>
<td>r</td>
</tr>
<tr>
<td>Reserve_Name</td>
<td>r, s</td>
</tr>
<tr>
<td>Resource_List</td>
<td>r, s</td>
</tr>
<tr>
<td>User_List</td>
<td>r</td>
</tr>
<tr>
<td><strong>Read-only attributes</strong></td>
<td></td>
</tr>
<tr>
<td>ctime</td>
<td>r</td>
</tr>
<tr>
<td>hashname</td>
<td>r</td>
</tr>
<tr>
<td>interactive</td>
<td>r, s</td>
</tr>
<tr>
<td>mtime</td>
<td>r</td>
</tr>
<tr>
<td>Queue</td>
<td>r</td>
</tr>
</tbody>
</table>
6.13.5.4 Using Resources in Hooks

All hooks can read, but not set, all job, vnode, server, queue, and reservation resources via pbs.server().job(), pbs.server().vnode(), pbs.server().queue(), etc. However, the resources that can be read or set via pbs.event() vary by hook. In the table below, if we say that a hook can read or set a group of resources, for example the server’s resources_available attribute, that means that the hook can read or set all of the resources for that group. Also, the queue

<table>
<thead>
<tr>
<th>Reservation Attribute</th>
<th>Readable, Settable</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserve_count</td>
<td>r</td>
</tr>
<tr>
<td>reserve_duration</td>
<td>r, s</td>
</tr>
<tr>
<td>reserve_end</td>
<td>r, s</td>
</tr>
<tr>
<td>reserve_index</td>
<td>r</td>
</tr>
<tr>
<td>reserve_ID</td>
<td>r</td>
</tr>
<tr>
<td>Reserve_Owner</td>
<td>r</td>
</tr>
<tr>
<td>reserve_retry</td>
<td>r</td>
</tr>
<tr>
<td>reserve_rule</td>
<td>r, s</td>
</tr>
<tr>
<td>reserve_start</td>
<td>r, s</td>
</tr>
<tr>
<td>reserve_state</td>
<td>r</td>
</tr>
<tr>
<td>reserve_substate</td>
<td>r</td>
</tr>
<tr>
<td>reserve_type</td>
<td>r</td>
</tr>
<tr>
<td>resv_nodes</td>
<td>r</td>
</tr>
<tr>
<td>server</td>
<td>r, s</td>
</tr>
<tr>
<td>Variable_List</td>
<td>r, s</td>
</tr>
</tbody>
</table>

Table 6-5: Reservation Attributes Readable & Settable via Events
that a resvsub hook operates on is the reservation queue. The following table shows the job, vnode, server, queue, and reservation resources that can be read or set in each type of hook, when retrieving the object through an event:

Table 6-6: Resources Readable & Settable by Hooks via Events

<table>
<thead>
<tr>
<th>Group of Resources</th>
<th>queuejob</th>
<th>modifyjob (before run)</th>
<th>movejob</th>
<th>runjob</th>
<th>resvsub</th>
<th>execjob_begin</th>
<th>execjob_prologue</th>
<th>execjob_end</th>
<th>execjob_epilogue</th>
<th>execjob_preterm</th>
<th>exechost_periodic</th>
<th>provision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Resource_List (restrictions apply; see 6.13.5.4.i)</td>
<td></td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Job resources_used</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Vnode resources_available</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>---</td>
</tr>
<tr>
<td>Vnode resources_assigned</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Server resources_available</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Server resources_assigned</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Server resources_default</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Server resources_max</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Queue resources_available</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Queue resources_assigned</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Queue resources_default</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Queue resources_max</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Queue resources_min</td>
<td>---</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Reservation Resource_List</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>r, s</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

PBS Professional 12.2 Administrator’s Guide  467
To summarize Table 6-6, “Resources Readable & Settable by Hooks via Events,” on page 467:

- Pre-execution event hooks can modify only the resources requested by a job. Execution event hooks can modify only the resources used by a job.
- The resvsub event hook can modify the resource arguments to pbs_rsub.
- Before a job runs, a modify job event hook can change any resource requested by the job, except for per-process limits such as pcput and pmem. While a job runs, a modifyjob event hook can change only cput and walltime.
- Execution event hooks can modify the resources_available belonging to the vnode on which the hook is running, and on other vnodes managed by the same MoM. Pre-execution event hooks cannot modify these resources.
- An execjob_begin, execjob_prologue, execjob_preterm, or execjob_epilogue event hook can set any resource used by the job, and any resources_available at the job’s vnodes.
- An execjob_end event hook can set any resources_available at the job’s vnodes.
- An execjob_epilogue event hook can capture and use the job’s exit status.
- An exechost_periodic event hook can set any resources_available at any vnode managed by the local MoM.

The built-in PBS resources are represented as Python dictionaries, where the resource names are the dictionary keys. These resources are listed in “Resources” on page 297 of the PBS Professional Reference Guide.

You can read a resource through objects such as the Server, the event that triggered the hook, or the vnode to which a resource belongs. For example:

```
pbs.server().resources_available["<resource name>"]
pbs.event().job.Resource_List["<resource name>"]
pbs.event().vnode_list[<vnode name>].resources_available["ncpus"]
```

The resource name must be in quotes.

Example: Get the number of CPUs in a job’s Resource_List attribute:

```
ncpus=pbs.event().job.Resource_List["ncpus"]
```

An instance $R$ of a job resource can be set as follows:

```
R["<resource name>"] = <resource value>
```

For example:

```
pbs.event().job.Resource_List["mem"] = 8gb
pbs.event().vnode_list[<vnode name>].resources_available["ncpus"] = 2
```
### 6.13.5.4.i  Restrictions on Using Job’s Resource_List Attribute by Hooks

Using `pbs.event()`, some execution event hooks shown in Table 6-6, “Resources Readable & Settable by Hooks via Events,” on page 467 have access to some built-in resources.

When a `runjob` hook rejects a job, the hook can modify any member of the job’s `Resource_List` attribute except for `nodect`. When a `runjob` hook accepts a job, it can modify only certain members. The following table shows the built-in members of the job’s `Resource_List` attribute that can be read or set in each type of hook, when retrieving the object through an event:

#### Table 6-7: Resources Readable & Settable by Hooks via Events

<table>
<thead>
<tr>
<th>Resource_List Member</th>
<th>queuejob</th>
<th>modifyjob (before run)</th>
<th>movejob</th>
<th>runjob (on reject)</th>
<th>runjob (on accept)</th>
<th>resvjob</th>
<th>execjob_begin</th>
<th>execjob_prologue</th>
<th>execjob_epilogue</th>
<th>execjob_preterm</th>
<th>exechost_periodic</th>
<th>provision</th>
</tr>
</thead>
<tbody>
<tr>
<td>accelerator</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
<tr>
<td>accelerator_memory</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
<tr>
<td>accelerator_model</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
<tr>
<td>aoe</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
<tr>
<td>arch</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
<tr>
<td>cput</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
<tr>
<td>exec_vnode</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
<tr>
<td>file</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
<tr>
<td>host</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
<tr>
<td>max_walltime</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
<tr>
<td>mem</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
<tr>
<td>min_walltime</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
<tr>
<td>mpiprocs</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>r, s</td>
</tr>
</tbody>
</table>
### Table 6-7: Resources Readable & Settable by Hooks via Events

<table>
<thead>
<tr>
<th>Resource_List Member</th>
<th>queuejob</th>
<th>modifyjob (before run)</th>
<th>movejob</th>
<th>runjob (on reject)</th>
<th>runjob (on accept)</th>
<th>resvsub</th>
<th>execjob_begin</th>
<th>execjob_prologue</th>
<th>execjob_end</th>
<th>execjob_epilogue</th>
<th>execjob_preterm</th>
<th>exechost_periodic</th>
<th>provision</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpparch</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>mppdepth</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>mpghost</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>mpplabels</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>mpmem</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>mppnodes</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>mppnpnn</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>mppwidth</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>naccelerators</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>nchunk</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>ncpus</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>netwins</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>nice</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>nodect</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>nodes</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>ompthreads</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>pcpput</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>pmem</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>pmem</td>
<td>r, s</td>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
Using `pbs.event()`, execution event hooks and periodic hooks cannot read or set custom resources in the job’s `Resource_List` attribute. This includes custom resources for the Cray; see “Custom Cray Resources” on page 308 of the PBS Professional Reference Guide. Instead, read these values through `pbs.server()`. For example:

```python
j = pbs.event().job
sj = pbs.server().job(j.id)
custres_val = sj.Resource_List["<name of custom resource>"]
```
6.13.6  Using select and place in Hooks

All hooks can read, but not set, a job’s select and place statements via `pbs.server().job()`, `pbs.server().vnode()`, `pbs.server().queue()`, etc. The following table shows the type of hook that can read or set a job’s select and place statements, when retrieving the object through an event:

Table 6-8: Hooks that Can Read & Set Job select and place Statements via Events

<table>
<thead>
<tr>
<th>Job place statement</th>
<th>queuejob</th>
<th>modifyjob (before run)</th>
<th>movejob</th>
<th>runjob (on reject)</th>
<th>runjob (on accept)</th>
<th>resvjob</th>
<th>execjob_begin</th>
<th>execjob_prologue</th>
<th>execjob_end</th>
<th>execjob_epilogue</th>
<th>execjob_preterm</th>
<th>exechost_periodic</th>
<th>provision</th>
</tr>
</thead>
<tbody>
<tr>
<td>r, s</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>Job select statement</td>
<td>r, s</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>r</td>
</tr>
</tbody>
</table>

6.13.6.1  How to Set select and place in Hooks

You must use the associated creation method to instantiate an object of the correct type with the desired value, then assign the object to the job. Format:

```
job.place = pbs.place("[arrangement]:[sharing]:[group]")
job.select = pbs.select("[N:\res=val[:res=val][+:N:\res=val]:res=val]...")
```

Example 6-13: Set a job’s select and place directives:

```python
jobB = pbs.event().job
jobB.place = pbs.place("pack:exclhost")
jobB.select = pbs.select("2:mem=2gb:ncpus=1+6:mem=8gb:ncpus=16")
```

See "`pbs.select("[N:\res=val[:res=val][+:N:\res=val]:res=val]...")` on page 533" and "`pbs.place("[arrangement]:[sharing]:[group]")` on page 532."
6.13.7 PBS Interface Objects

The PBS interface contains different kinds of objects:

- Constant objects to represent event types, states, log event classes, queue types, and exceptions. These cannot be changed.
- Objects to represent PBS entities, e.g. jobs, server, queues, vnodes, reservations, events, log messages, etc.
- Objects to represent job, server, vnode, queue, and reservation attributes.
- Objects to represent arguments to PBS commands, PBS version information, etc.

6.13.7.1 Table of PBS Interface Objects

PBS provides a set of interface objects for use in hooks. These objects are all `pbs.*` objects, and are inherited by other objects. For example, `pbs.checkpoint` appears as a job object member as `pbs.job.checkpoint`. The following table lists all of the PBS objects in alphabetical order. Each of these objects is described in detail later in the chapter.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.acl</td>
<td>Represents a PBS ACL. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.args</td>
<td>Represents a space-separated list of PBS arguments to commands like <code>qsub</code>, <code>qdel</code>. See “Other Global Creation Methods” on page 534.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.BadAttributeValueError</td>
<td>Raised when setting the member value of a <code>pbs.*</code> object and the value given is invalid. See “Exceptions” on page 484.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.BadAttributeValueTypeError</td>
<td>Raised when setting the member value of a <code>pbs.*</code> object and the value type is invalid. See “Exceptions” on page 484.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.BadResourceValueError</td>
<td>Raised when setting the resource value of a <code>pbs.*</code> object and the value given is invalid. See “Exceptions” on page 484.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
Table 6-9: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.BadResourceValueTypeError</td>
<td>Raised when setting the resource value of a pbs.* object and the value type is invalid. See “Exceptions” on page 484.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.checkpoint</td>
<td>Represents a job's checkpoint attribute. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.depend</td>
<td>Represents a job's dependency attribute. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.duration</td>
<td>Represents a time interval. See “Global Attribute Creation Methods” on page 529.</td>
<td>--</td>
</tr>
<tr>
<td>pbs.email_list</td>
<td>Represents the set of users to whom mail may be sent. Example: Job's Mail_Users attribute. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.event</td>
<td>Represents a PBS event. See “Event Objects” on page 492.</td>
<td>--</td>
</tr>
<tr>
<td>pbs.EventIncompatibleError</td>
<td>Raised when referencing a nonexistent member in pbs.event(). See “Exceptions” on page 484.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.EXECHOST_PERIODIC</td>
<td>Type for an exechost_periodic hook event.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.EXECJOB_BEGIN</td>
<td>Type for an execjob_begin hook event.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.EXECJOB_END</td>
<td>Type for an execjob_end hook event.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
Table 6-9: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pbs.EXECJOB_EPILOGUE</code></td>
<td>Type for an execjob_epilogue hook event.</td>
<td>Constant</td>
</tr>
<tr>
<td><code>pbs.EXECJOB_PRETERM</code></td>
<td>Type for an execjob_preterm hook event.</td>
<td>Constant</td>
</tr>
<tr>
<td><code>pbs.EXECJOB_PROLOGUE</code></td>
<td>Type for an execjob_prologue hook event.</td>
<td>Constant</td>
</tr>
<tr>
<td><code>pbs.exec_host</code></td>
<td>Represents a job’s exec_host attribute. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td><code>pbs.exec_vnode</code></td>
<td>Represents a job’s exec_vnode attribute. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td><code>pbs.group_list</code></td>
<td>Represents a list of group names. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td><code>pbs.hold_types</code></td>
<td>Represents the Hold_Types attribute of a job. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td><code>pbs.job</code></td>
<td>Represents a PBS job. See “Job Objects” on page 519.</td>
<td>--</td>
</tr>
<tr>
<td><code>pbs.job_sort_formula</code></td>
<td>Represents the job_sort_formula server attribute. See “pbs.job_sort_formula(&quot;&lt;formula string&gt;&quot;)” on page 531.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td><code>pbs.JOB_STATE_BEGUN</code></td>
<td>Job arrays only. Job array has started. See “Job State Objects” on page 488.</td>
<td>Constant</td>
</tr>
<tr>
<td><code>pbs.JOB_STATE_EXITING</code></td>
<td>Job is exiting after having run. See “Job State Objects” on page 488.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
### Hooks

#### Table 6-9: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.JOB_STATE_EXPIRED</td>
<td>Subjobs only. Subjob is finished (expired.) See “Job State Objects” on page 488.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_FINISHED</td>
<td>Job is finished: job executed successfully, job was terminated while running, job execution failed, or job was deleted before execution. See &quot;Job State Objects&quot; on page 488.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_HELD</td>
<td>Job is held. See “Job State Objects” on page 488.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_MOVED</td>
<td>Job has been moved to another server. See &quot;Job State Objects” on page 488.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_QUEUED</td>
<td>Job is queued, eligible to run or be routed. See “Job State Objects” on page 488.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_RUNNING</td>
<td>Job is running. See “Job State Objects” on page 488.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_SUSPEND</td>
<td>Job is suspended by server. See “Job State Objects” on page 488.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_SUSPEND_USERACTIVE</td>
<td>Job is suspended due to workstation becoming busy. See “Job State Objects” on page 488.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_TRANSIT</td>
<td>Job is in transit. See “Job State Objects” on page 488.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_WAITING</td>
<td>Job is waiting for its requested execution time to be reached, or the job’s stagein request has failed. See “Job State Objects” on page 488.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
### Table 6-9: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.join_path</td>
<td>Represents the job’s Join_Path attribute. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.keep_files</td>
<td>Represents the Keep_Files job attribute. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.license_count</td>
<td>Represents a set of licensing-related counters. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.LOG_DEBUG</td>
<td>Log event class 0x0004. See “Message Log Event Class Objects” on page 487.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.LOG_ERROR</td>
<td>Log event class 0x0004. See “Message Log Event Class Objects” on page 487.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.LOG_WARNING</td>
<td>Log event class 0x0004. See “Message Log Event Class Objects” on page 487.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.mail_points</td>
<td>Represents the Mail_Points attribute of a job. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.MODIFYJOB</td>
<td>The modifyjob hook event type. Triggered by qalter or pbs_alterjob() API call. See “Event Types” on page 496.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.MOVEJOB</td>
<td>The movejob hook event type. Triggered by qmove or pbs_movejob() API call. See “Event Types” on page 496.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_BUSY</td>
<td>Represents busy vnode state. See section 6.13.8.8, &quot;Vnode State Objects&quot;, on page 490.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
Table 6-9: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.ND_DOWN</td>
<td>Represents down vnode state. See section 6.13.8.8, “Vnode State Objects”, on page 490.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_FREE</td>
<td>Represents free vnode state. See section 6.13.8.8, “Vnode State Objects”, on page 490.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_GLOBUS</td>
<td>Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus. No longer used. Represents globus value for vnode ntype attribute. See section 6.13.8.9, “Vnode Type Objects”, on page 491.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_JOBBUSY</td>
<td>Represents job-busy vnode state. See section 6.13.8.8, “Vnode State Objects”, on page 490.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_JOB_EXCLUSIVE</td>
<td>Represents job-exclusive vnode state. See section 6.13.8.8, “Vnode State Objects”, on page 490.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
## Table 6-9: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.ND_OFFLINE</td>
<td>Represents \textit{offline} vnode state. See section 6.13.8.8, “Vnode State Objects”, on page 490.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_PBS</td>
<td>Represents \textit{pbs} value for vnode \texttt{n}\texttt{t}\texttt{p}\texttt{e} attribute. See section 6.13.8.9, “Vnode Type Objects”, on page 491.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_PROV</td>
<td>Represents \textit{provisioning} vnode state. See section 6.13.8.8, “Vnode State Objects”, on page 490.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_RESV_EXCLUSIVE</td>
<td>Represents \textit{resv-exclusive} vnode state. See section 6.13.8.8, “Vnode State Objects”, on page 490.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_STALE</td>
<td>Represents \textit{stale} vnode state. See section 6.13.8.8, “Vnode State Objects”, on page 490.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_STATE_UNKNOWN</td>
<td>Represents \textit{state-unknown, down} vnode state. See section 6.13.8.8, “Vnode State Objects”, on page 490.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_UNRESOLVABLE</td>
<td>Represents \textit{unresolvable} vnode state. See section 6.13.8.8, “Vnode State Objects”, on page 490.</td>
<td></td>
</tr>
<tr>
<td>pbs.ND_WAIT_PROV</td>
<td>Represents \textit{wait-provisioning} vnode state. See section 6.13.8.8, “Vnode State Objects”, on page 490.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.node_group_key</td>
<td>Represents the node_group_key attribute. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
</tbody>
</table>
## Hooks

### Table 6-9: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.path_list</td>
<td>Represents a list of pathnames. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.place</td>
<td>Represents the place specification when submitting a job. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.QTYPE_EXECUTION</td>
<td>Represents execution value for queue_type queue attribute. See “Queue Type Objects” on page 487.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.QTYPE_ROUTE</td>
<td>Represents route value for queue_type queue attribute. See “Queue Type Objects” on page 487.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.queue</td>
<td>Represents a PBS queue. See “Queue Objects” on page 518.</td>
<td>--</td>
</tr>
<tr>
<td>pbs.QUEUEJOB</td>
<td>The queuejob hook event type. Triggered by qsub or pbs_submit() API call. See “Event Types” on page 496.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.range</td>
<td>Represents a range of numbers referring to job array indices. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.resv</td>
<td>Represents a PBS reservation. See “Reservation Objects” on page 526.</td>
<td>--</td>
</tr>
<tr>
<td>pbs.RESVSUB</td>
<td>The resvsub hook event type. Triggered by pbs_rsub or pbs_submitresv() API call. See “Event Types” on page 496.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.RESV_STATE_BEING_DELETED</td>
<td>The reservation state RESV_BEING_DELETED. See “Reservation State Objects” on page 489.</td>
<td>Constant</td>
</tr>
</tbody>
</table>

480  PBS  Professional 12.2  Administrator’s  Guide
## Hooks

### Chapter 6

#### Table 6-9: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pbs.RESV_STATE_CONFIRMED</code></td>
<td>The reservation state <code>RESV_CONFIRMED</code>. See “Reservation State Objects” on page 489.</td>
<td>Constant</td>
</tr>
<tr>
<td><code>pbs.RESV_STATE_DEGRADED</code></td>
<td>The reservation state <code>RESV_DEGRADED</code>. See “Reservation State Objects” on page 489.</td>
<td>Constant</td>
</tr>
<tr>
<td><code>pbs.RESV_STATE_DELETED</code></td>
<td>The reservation state <code>RESV_DELETED</code>. See “Reservation State Objects” on page 489.</td>
<td>Constant</td>
</tr>
<tr>
<td><code>pbs.RESV_STATE_DELETING_JOBS</code></td>
<td>The reservation state <code>RESV_DELETING_JOBS</code>. See “Reservation State Objects” on page 489.</td>
<td>Constant</td>
</tr>
<tr>
<td><code>pbs.RESV_STATE_FINISHED</code></td>
<td>The reservation state <code>RESV_FINISHED</code>. See “Reservation State Objects” on page 489.</td>
<td>Constant</td>
</tr>
<tr>
<td><code>pbs.RESV_STATE_NONE</code></td>
<td>The reservation state <code>RESV_NONE</code>. See “Reservation State Objects” on page 489.</td>
<td>Constant</td>
</tr>
<tr>
<td><code>pbs.RESV_STATE_RUNNING</code></td>
<td>The reservation state <code>RESV_RUNNING</code>. See “Reservation State Objects” on page 489.</td>
<td>Constant</td>
</tr>
<tr>
<td><code>pbs.RESV_STATE_TIME_TO_RUN</code></td>
<td>The reservation state <code>RESV_TIME_TO_RUN</code>. See “Reservation State Objects” on page 489.</td>
<td>Constant</td>
</tr>
<tr>
<td><code>pbs.RESV_STATE_UNCONFIRMED</code></td>
<td>The reservation state <code>RESV_UNCONFIRMED</code>. See “Reservation State Objects” on page 489.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
# Hooks

## Table 6-9: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.RESV_STATE_WAIT</td>
<td>The reservation state <strong>RESV_WAIT</strong>. See “Reservation State Objects” on page 489.</td>
<td>Constant</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pbs.route_destinations</td>
<td>Represents route_destinations queue attribute. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pbs.RUNJOB</td>
<td>The runjob hook event type. Triggered by qrun or pbs_runjob() API call. See “Event Types” on page 496.</td>
<td>Constant</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pbs.select</td>
<td>Represents the select specification when submitting a job. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pbs.server</td>
<td>Represents the local PBS server. See “Server Objects” on page 515.</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pbs.size</td>
<td>Represents a PBS size type. See “Global Attribute Creation Methods” on page 529.</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pbs.software</td>
<td>Represents a site-dependent software specification resource. See “Other Global Creation Methods” on page 534.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pbs.staging_list</td>
<td>Represents a list of file stagein or stageout parameters. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pbs.state_count</td>
<td>Represents a set of job-related state counters. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pbs.SV_STATE_ACTIVE</td>
<td>Server state is <strong>Scheduling</strong>. See “PBS Server State Objects” on page 487.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
## Table 6-9: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pbs.SV_STATE_HOT</strong></td>
<td>Server state is <em>Hot_Start</em>. See “PBS Server State Objects” on page 487.</td>
<td>Constant</td>
</tr>
<tr>
<td><strong>pbs.SV_STATE_IDLE</strong></td>
<td>Server state is <em>Idle</em>. See “PBS Server State Objects” on page 487.</td>
<td>Constant</td>
</tr>
<tr>
<td><strong>pbs.SV_STATE_SHUTDEL</strong></td>
<td>Server state is <em>Terminating, Delayed</em>. See “PBS Server State Objects” on page 487.</td>
<td>Constant</td>
</tr>
<tr>
<td><strong>pbs.SV_STATE_SHUTIMM</strong></td>
<td>Server state is <em>Terminating</em>. See “PBS Server State Objects” on page 487.</td>
<td>Constant</td>
</tr>
<tr>
<td><strong>pbs.SV_STATE_SHUTSIG</strong></td>
<td>Server state is <em>Terminating</em>. See “PBS Server State Objects” on page 487.</td>
<td>Constant</td>
</tr>
<tr>
<td><strong>pbs.UnsetAttributeNameError</strong></td>
<td>Raised when referencing a non-existent member name of a pbs.* object. See “Exceptions” on page 484.</td>
<td>Constant</td>
</tr>
<tr>
<td><strong>pbs.UnsetResourceNameError</strong></td>
<td>Raised when referencing a non-existent resource name of a pbs.* object. See “Exceptions” on page 484.</td>
<td>Constant</td>
</tr>
<tr>
<td><strong>pbs.user_list</strong></td>
<td>Represents a list of usernames. See “Global Attribute Creation Methods” on page 529.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td><strong>pbs.vchunk</strong></td>
<td>Represents a chunk. See section 6.13.14, “Chunk Objects”, on page 526.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td><strong>pbs.version</strong></td>
<td>Represents version information for PBS. See “Other Global Creation Methods” on page 534.</td>
<td>Formatted input string</td>
</tr>
</tbody>
</table>
Table 6-9: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemExit</td>
<td>Raised when accepting or rejecting an action. See “Exceptions” on page 484.</td>
</tr>
</tbody>
</table>

### 6.13.7.2 Exceptions

The following exceptions may be raised when using the pbs.* objects:

Table 6-10: Exceptions Raised When Using pbs.* Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.BadAttributeValueError</td>
<td>Raised when setting member value of a pbs.* object to an invalid value.</td>
</tr>
<tr>
<td>pbs.BadAttributeValueTypeError</td>
<td>Raised when setting member value of a pbs.* object to an invalid type.</td>
</tr>
<tr>
<td>pbs.BadResourceValueError</td>
<td>Raised when setting resource value of a pbs.* object to an invalid value.</td>
</tr>
<tr>
<td>pbs.BadResourceValueTypeError</td>
<td>Raised when setting resource value of a pbs.* object to an invalid type.</td>
</tr>
<tr>
<td>pbs.EventIncompatibleError</td>
<td>Raised when referencing a nonexistent member in pbs.event. Example: calling pbs.event().resv for pbs.event().type of pbs.QUEUEJOB</td>
</tr>
<tr>
<td>pbs.UnsetAttributeNameError</td>
<td>Raised when referencing a non-existent member name of a pbs.* object.</td>
</tr>
<tr>
<td>pbs.UnsetResourceNameError</td>
<td>Raised when referencing a non-existent resource name of a pbs.* object.</td>
</tr>
<tr>
<td>SystemExit</td>
<td>1. Raised when pbs.event().reject() terminates hook execution.</td>
</tr>
<tr>
<td></td>
<td>2. Raised when pbs.event().accept() terminates hook execution.</td>
</tr>
</tbody>
</table>
6.13.7.2.1 Unhandled Exceptions

If a hook encounters an unhandled exception:

- PBS rejects the corresponding action. The command that initiates the action results in the following message in stderr:
  "<command_name>: request rejected as filter hook <hook_name> encountered an exception. Please inform Admin"

- The following message appears in the appropriate PBS daemon log, logged under 0x0100 event class:
  "<request type> hook <hook_name> encountered an exception, request rejected"

- The job is left unmodified.

The following statements will cause an unhandled exception if they appear in a hook script as is:

- ZeroDivisionError exception raised:
  `val = 5/0`

- BadAttributeValueError exception raised; pbs.hold_types and strings don't mix:
  `pbs.event().Hold_Types = "z"`

- EventIncompatibleError exception raised for the following runjob event; runjob event has job attribute, not resv attribute:
  `r = pbs.event().resv`

6.13.8 Constant Objects

Constant objects are used to represent PBS elements such as event types, job, server, reservation, and vnode states, log event classes, queue and vnode types, and exceptions. These objects cannot be modified. When the PBS module is imported, the constant objects are imported.

6.13.8.1 Caveats for Constant Objects

Constant objects are integers. When you want to compare the value of a member to a constant object, you must get integer representation of the member value first. For an explanation and example, see section 6.15.7.8, “Comparing Member to Constant Value or Constant Object”, on page 557. Here is a quick example:

```python
    e = pbs.event()
    if int(str(e.job.job_state)) == pbs.JOB_STATE_RUNNING :
        e.accept
```
6.13.8.2  Event Type Objects

The *pbs.event.type* event member represents the type of the event, for example, *queuejob* or *movejob*. This table lists the constant objects that represent event types. See section 6.13.9, “Event Objects”, on page 492.

Table 6-11: Event Type Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Event Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.QUEUEJOB</td>
<td>queuejob</td>
<td>qsub or pbs_submit()</td>
</tr>
<tr>
<td>pbs.MODIFYJOB</td>
<td>modifyjob</td>
<td>qalter or pbs_alterjob()</td>
</tr>
<tr>
<td>pbs.MOVEJOB</td>
<td>movejob</td>
<td>qmove or pbs_movejob()</td>
</tr>
<tr>
<td>pbs.RUNJOB</td>
<td>runjob</td>
<td>qrun or pbs_runjob()</td>
</tr>
<tr>
<td>pbs.EXECJOB_BEGIN</td>
<td>execjob_begin</td>
<td>Execution host receives job</td>
</tr>
<tr>
<td>pbs.EXECJOB_PROLOGUE</td>
<td>execjob_prologue</td>
<td>Just before executing top-level shell or cmd process of job</td>
</tr>
<tr>
<td>pbs.EXECJOB_EPILOGUE</td>
<td>execjob_epilogue</td>
<td>Just after job processes have been killed</td>
</tr>
<tr>
<td>pbs.EXECJOB_END</td>
<td>execjob_end</td>
<td>After all job processes have been cleaned up</td>
</tr>
<tr>
<td>pbs.EXECJOB_PRETERM</td>
<td>execjob_preterm</td>
<td>When job receives terminate signal</td>
</tr>
<tr>
<td>pbs.EXECHOST_PERIODIC</td>
<td>exechost_periodic</td>
<td>Not triggered by action; triggered at specified interval</td>
</tr>
<tr>
<td>pbs.RESVSUB</td>
<td>resvsub</td>
<td>pbs_rsub or pbs_submitresv()</td>
</tr>
<tr>
<td>pbs.PROVISION</td>
<td>provision</td>
<td>Provisioning a vnode</td>
</tr>
</tbody>
</table>
6.13.8.3 Message Log Event Class Objects

You can use the following objects to indicate log event class when placing messages in the server logs. Log levels are shown in hexadecimal. The messages are logged at the "Administrative Events" (0x0004) event class.

<table>
<thead>
<tr>
<th>Object</th>
<th>Log Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.LOG_WARNING</td>
<td>0x0004</td>
</tr>
<tr>
<td>pbs.LOG_ERROR</td>
<td>0x0004</td>
</tr>
<tr>
<td>pbs.LOG_DEBUG</td>
<td>0x0004</td>
</tr>
</tbody>
</table>

6.13.8.4 Queue Type Objects

The pbs.queue.queue_type member represents the type of the queue. It can take on the following values:

<table>
<thead>
<tr>
<th>Object</th>
<th>Queue Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.QTYPE_EXECUTION</td>
<td>Execution</td>
</tr>
<tr>
<td>pbs.QTYPE_ROUTE</td>
<td>Route</td>
</tr>
</tbody>
</table>

6.13.8.5 PBS Server State Objects

The pbs.server.server_state member represents the state of the server. It can take on the following values:

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.SV_STATE_IDLE</td>
<td>Idle</td>
</tr>
<tr>
<td>pbs.SV_STATE_ACTIVE</td>
<td>Scheduling</td>
</tr>
</tbody>
</table>
6.13.8.6 Job State Objects

The `pbs.job.job_state` member represents the job’s state. Use these constant objects to test the state of a job. When you perform this test, get the integer value of the `job_state` member (even though its type is `pbs.job_state`). For example:

```python
e = pbs.event()
if int(str(e.job.job_state)) == pbs.JOB_STATE_RUNNING :
e.accept
```

The `job_state` member can take on any of the values listed here:

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pbs.JOB_STATE_BEGUN</code></td>
<td><code>B</code></td>
<td>Job arrays only: job array has started</td>
</tr>
<tr>
<td><code>pbs.JOB_STATE_EXITING</code></td>
<td><code>E</code></td>
<td>Job is exiting after having run</td>
</tr>
<tr>
<td><code>pbs.JOB_STATE_EXPIRED</code></td>
<td><code>X</code></td>
<td>Subjobs only; subjob is finished (expired.)</td>
</tr>
<tr>
<td><code>pbs.JOB_STATE_FINISHED</code></td>
<td><code>F</code></td>
<td>Job is finished: job executed successfully, job was terminated while running, job execution failed, or job was deleted before execution</td>
</tr>
<tr>
<td><code>pbs.JOB_STATE_HELD</code></td>
<td><code>H</code></td>
<td>Job is held.</td>
</tr>
</tbody>
</table>
6.13.8.7 Reservation State Objects

The `pbs.resv.reserve_state` member represents the state of the reservation. It can take on the following values:

### Table 6-16: Reservation State Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.RESV_STATE_NONE</td>
<td>RESV_NONE</td>
<td></td>
</tr>
<tr>
<td>pbs.RESV_STATE_UNCONFIRMED</td>
<td>RESV_UNCONFIRMED</td>
<td></td>
</tr>
<tr>
<td>pbs.RESV_STATE_CONFIRMED</td>
<td>RESV_CONFOIRMED</td>
<td></td>
</tr>
<tr>
<td>pbs.RESV_STATE_WAIT</td>
<td>RESV_WAIT</td>
<td></td>
</tr>
</tbody>
</table>

### Table 6-15: Job State Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.JOB_STATE_MOVED</td>
<td>M</td>
<td>Job has been moved to another server</td>
</tr>
<tr>
<td>pbs.JOB_STATE_QUEUED</td>
<td>Q</td>
<td>Job is queued, eligible to run or be routed</td>
</tr>
<tr>
<td>pbs.JOB_STATE_RUNNING</td>
<td>R</td>
<td>Job is running</td>
</tr>
<tr>
<td>pbs.JOB_STATE_SUSPEND</td>
<td>S</td>
<td>Job is suspended by PBS so that a higher-priority job can run.</td>
</tr>
<tr>
<td>pbs.JOB_STATE_SUSPEND_USERACTIVE</td>
<td>U</td>
<td>Job is suspended due to workstation becoming busy</td>
</tr>
<tr>
<td>pbs.JOB_STATE_TRANSIT</td>
<td>T</td>
<td>Job is in transition (being moved to a new location)</td>
</tr>
<tr>
<td>pbs.JOB_STATE_WAITING</td>
<td>W</td>
<td>Job is waiting for its requested execution time to be reached, or the job’s specified stage in request has failed for some reason.</td>
</tr>
</tbody>
</table>
6.13.8.8 Vnode State Objects

The `pbs.vnode.state` member represents the state of the vnode. It can take on the following values:

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pbs.ND_BUSY</code></td>
<td>Represents <em>busy</em> vnode state</td>
</tr>
<tr>
<td><code>pbs.ND_DOWN</code></td>
<td>Represents <em>down</em> vnode state</td>
</tr>
<tr>
<td><code>pbs.ND_FREE</code></td>
<td>Represents <em>free</em> vnode state</td>
</tr>
<tr>
<td><code>pbs.ND_JOB_BUSY</code></td>
<td>Represents <em>job-busy</em> vnode state</td>
</tr>
<tr>
<td><code>pbs.ND_JOB_EXCLUSIVE</code></td>
<td>Represents <em>job-exclusive</em> vnode state</td>
</tr>
<tr>
<td><code>pbs.ND_OFFLINE</code></td>
<td>Represents <em>offline</em> vnode state</td>
</tr>
<tr>
<td><code>pbs.ND_PROV</code></td>
<td>Represents <em>provisioning</em> vnode state</td>
</tr>
<tr>
<td><code>pbs.ND_RESV_EXCLUSIVE</code></td>
<td>Represents <em>resv-exclusive</em> vnode state</td>
</tr>
<tr>
<td><code>pbs.ND_STALE</code></td>
<td>Represents <em>stale</em> vnode state</td>
</tr>
</tbody>
</table>
6.13.8.9 Vnode Type Objects

The pbs.vnode.ntype member represents the type of the vnode. It can take on the following values:

<table>
<thead>
<tr>
<th>Object</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.ND_GLOBUS</td>
<td>Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus. No longer used. Represents <em>globus</em> value for vnode ntype attribute</td>
</tr>
<tr>
<td>pbs.ND_PBS</td>
<td>Represents <em>pbs</em> value for vnode ntype attribute</td>
</tr>
</tbody>
</table>

6.13.8.10 Vnode Sharing Objects

The pbs.vnode.sharing member represents the vnode’s sharing attribute. It can take on the following values:

<table>
<thead>
<tr>
<th>Object</th>
<th>Sharing Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.ND_DEFAULT_EXCL</td>
<td>Represents <em>default_excl</em> vnode sharing attribute value</td>
</tr>
<tr>
<td>pbs.ND_DEFAULT_EXCLHOST</td>
<td>Represents <em>default_exclhost</em> vnode sharing attribute value</td>
</tr>
</tbody>
</table>
The event object represents the event that has triggered the hook. You can pass the object to the hook script, and use it in the script.

To create a PBS event object to represent the current event:

```plaintext
pbs.event()
```

The event object can be represented by this:

```plaintext
pbs.event
```

There are several types of events. Each type of event is triggered by a different occurrence, and each type has a corresponding hook type. Each type of event has access to different data, and can perform different operations. Some data and operations are common to all events.

In the following sections, we first describe the data and operations (members and methods) that are common to all events. After that, we list each type of event and describe the members and methods that are special to it.

Each type of event hook can read and set different job, vnode, and reservation attributes and resources. Each type of event can read different server and queue attributes and resources. We list which attributes and resources can be set for each event in section 6.13.5, “Using Attributes and Resources in Hooks”, on page 456.

### 6.13.9.1 Event Object Members for All Events

Event objects have different members depending on the type of event. The members for each type are described in the following sections. Some event members are common to all events.

---

**Table 6-19: Vnode Sharing Objects**

<table>
<thead>
<tr>
<th>Object</th>
<th>Sharing Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.ND_DEFAULT_SHARED</td>
<td>Represents default_shared vnode sharing attribute value</td>
</tr>
<tr>
<td>pbs.ND_FORCE_EXCL</td>
<td>Represents force_excl vnode sharing attribute value</td>
</tr>
<tr>
<td>pbs.ND_FORCE_EXCLHOST</td>
<td>Represents force_exclhost vnode sharing attribute value</td>
</tr>
<tr>
<td>pbs.ND_IGNORE_EXCL</td>
<td>Represents ignore_excl vnode sharing attribute value</td>
</tr>
</tbody>
</table>
An object returned by `pbs.event()` has the following members:

- `pbs.event().hook_alarm`
  Number of seconds hook is allowed to run. Represents the `alarm` hook attribute.
  Type: `int`

- `pbs.event().hook_name`
  Name of the hook being executed.
  Type: `str`

- `pbs.event().hook_type`
  The type of the hook. The only valid value is “site”. Represents the `Type` hook attribute.
  Type: `str`

- `pbs.event().requestor`
  The requestor of the event.
  PBS daemons can request actions. If a daemon requests an action, the requestor member contains one of "PBS_Server", "Scheduler", or "pbs_mom". If the requestor is root, the member contains “root”. For Windows systems, if the requestor is the administrator, the member contains the account name of the administrator.
  Type: `str`

- `pbs.event().requestor_host`
  The name of the host from which the event was requested.
  Type: `str`

- `pbs.event().type`
  The event type, for example, `queuejob` or `movejob`. Valid values: one of the PBS event type constants listed in section 6.13.9.5, “Event Types”, on page 496.
  Type: constant event object

### 6.13.9.2 Event Object Members for Execution and Periodic Events

#### 6.13.9.2.i The Vnode List Member

Execution event hooks and periodic event hooks have access to the list of vnodes managed by the MoM where the hook is running. This means that if one MoM is managing multiple vnodes, a hook run by that MoM has access to all of the vnodes managed by the MoM. However, for a hook running where its MoM manages just one vnode, that hook has access only to its own vnode.
A periodic hook has access to all of the information about every vnode. Execution hooks have access to attributes and resources for the vnodes assigned to the job associated with the instance of the hook, where those vnodes are also managed by the hook’s MoM.

The execution event member that represents a list of vnodes looks like this:

```python
pbs.event().vnode_list[]
```

You can set the attributes and resources for a particular vnode as follows:

```python
pbs.event().vnode_list[vnode name].<attribute> = <value>
pbs.event().vnode_list[vnode name].resources_available["<resource name>"] = <value>
```

Resource names and string values must be quoted.

Some examples:

```python
pbs.event().vnode_list[vnode name].pcpus = 5
pbs.event().vnode_list[vnode name].resources_available["ncpus"] = 3
pbs.event().vnode_list[vnode name].resources_available["mem"] = pbs.size("100gb")
pbs.event().vnode_list[vnode name].arch = "linux"
pbs.event().vnode_list[vnode name].state = pbs.ND_OFFLINE
pbs.event().vnode_list[vnode name].sharing = pbs.ND_FORCE_EXCL
```

If you want to use a non-periodic execution event hook to manipulate a vnode that is not assigned to the job, but is still managed by the hook’s MoM, you must first instantiate the object for that vnode with the name of the new vnode:

```python
pbs.event().vnode_list[new vnode] = pbs.vnode(<new vnode name>)
```

Once you have instantiated your new vnode (which must still be managed by your hook’s MoM), you can operate on it as shown above.

You can find the other vnodes managed by the hook’s MoM by querying the server for its list of vnodes using `pbs.server().vnodes()`, then looking in the resulting list of vnodes for a match to the output of `pbs.get_local_nodename()`, which returns the name of the local natural vnode.

You cannot modify a vnode that is managed by a different MoM from where the hook is running. If you try to do this, the following error message appears in the server’s log at log event class 0x0100:

```
"<node_host_name>; Not allowed to update <vnode name>, as it is owned by a different mom"
```
A hook that runs as "pbsuser" (execjob_prologue, execjob_epilogue, execjob_preterm) is not allowed to manipulate pbs.event().vnode_list[], unless the executing user is a PBS Manager or Operator. If a hook running as an unprivileged user tries to change pbs.event().vnode_list[], the following error message appears in the server’s log at log event class 0x0100:

"<node_host_name>; Not allowed to update vnodes or to request scheduler restart cycle, if run as a non-manager/operator user"

### 6.13.9.3 Event Object Methods for All Events

```python
pbs.event().reject([<error message>],<error code>)
```

Terminates hook execution and instructs PBS to not perform the associated action. If the `<message>` argument is given, it is shown in the appropriate PBS daemon log, and in the stderr of the PBS command that caused this event to take place.

By default, pbs.event().reject() returns 255. To return an error code other than 255, specify a value between 2 and 255 in the optional `<error code>`.

```python
pbs.event().accept()
```

Terminates hook execution and causes PBS to perform the associated action.

### 6.13.9.3.i Event Object Method Caveats

`pbs.event().accept()` terminates hook execution by throwing a `SystemExit` exception. So if hook content appears in a `try...except` clause that has no arguments to the `except` clause, always add the following to treat `SystemExit` as a normal occurrence:

```python
except SystemExit:
    pass
```

See section 6.15.8.1, "Treating `SystemExit` as a Normal Occurrence", on page 558.

### 6.13.9.4 Event Object Methods for Some Events

#### 6.13.9.4.i Method to Reboot Host

Execution event hooks and periodic event hooks have a method to reboot a host, called `pbs.reboot()`:

```python
pbs.reboot([<command>])
```

This stops hook execution, so that remaining lines in the hook script are not executed, and starts the tasks that would normally begin after the hook is finished, such as flagging the current host to be rebooted. The MoM logs show the following:

- `<hook name> requested for host to be rebooted`
We recommend that before calling `pbs.reboot()`, you set any vnodes managed by this MoM offline, and requeue the current job, if this hook is not an `exechost_periodic` hook. For example:

```python
for v in pbs.event().vnode_list.keys():
    pbs.event().vnode_list[v].state = pbs.ND_OFFLINE
    pbs.event().vnode_list[v].comment = "Mom host rebooting"
    pbs.event().job.rerun()
    pbs.reboot()
```

The effect of the call to `pbs.reboot()` is not instantaneous. The reboot happens after the hook executes, and after any of the other actions such as `pbs.event().job.rerun()`, `pbs.event().delete()`, and `pbs.event().vnode_list[]` take effect.

A hook with its `user` attribute set to `pbsuser` cannot successfully invoke `pbs.reboot()`, even if the owner is a PBS Manager or Operator. If this is attempted, the host is not rebooted, and the following message appears at log event class 0x0100 in the MoM logs:

```
<hook_name>; Not allowed to issue reboot if run as user.
```

The `<command>` is an optional argument. It is a Python `str` which is executed instead of the reboot command that is the default for the system. For example:

```python
pbs.reboot("/usr/local/bin/my_reboot -s 10 -c 'going down in 10'")
```

The specified `<command>` is executed in a shell on Linux/UNIX or via `cmd` on Windows.

**6.13.9.5 Event Types**

The type of the event. This is the object that is returned by `pbs.event().type`.

Chapter 6  
**Hooks**

---

496  
PBS Professional 12.2 Administrator’s Guide
The following table summarizes the event types, their triggers, and where they run, and gives a pointer to a complete description of the associated hook:

### Table 6-20: Event Type Objects

<table>
<thead>
<tr>
<th>Type</th>
<th>Trigger</th>
<th>Where Run</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queuejob</td>
<td>Triggered by qsub and the pbs_submit() API call. Not triggered by requeueing a job (qre-run) or on node_fail_requeue, when a job is discarded by the MoM because the execution host went down. A queuejob hook is executed after all processing of qsub input, and just before the job is queued.</td>
<td>At server</td>
<td>See section 6.13.9.5.i, “queuejob: Event when Job is Queued”, on page 500.</td>
</tr>
<tr>
<td>modifyjob</td>
<td>Triggered by qalter and the pbs_alterjob() API call. A modifyjob hook is executed after all processing of qalter input, and just before the job's attributes are modified.</td>
<td>At server</td>
<td>See section 6.13.9.5.ii, “modifyjob: Event when Job is Altered”, on page 500.</td>
</tr>
</tbody>
</table>
# Hooks

## Table 6-20: Event Type Objects

<table>
<thead>
<tr>
<th>Type</th>
<th>Trigger</th>
<th>Where Run</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>movejob</td>
<td>Triggered by qmove and the <code>pbs_movejob()</code> API call. Not triggered by <code>pbs_rsub -Wqmove=&lt;job ID&gt;</code>. A movejob hook is executed after qmove arguments are processed, but before a job is moved from one queue to another.</td>
<td>At server</td>
<td>See section 6.13.9.5.iii, “movejob: Event when Job is Moved”, on page 501.</td>
</tr>
<tr>
<td>runjob</td>
<td>Triggered by qrun and the <code>pbs_runjob()</code> API call. A runjob hook is executed just before a job is sent to an execution host.</td>
<td>At server</td>
<td>See section 6.13.9.5.iv, “runjob: Event Before Job is Received by MoM”, on page 502.</td>
</tr>
<tr>
<td>execjob_begin</td>
<td>An execjob_begin hook is executed when MoM receives the job, after any files or directories are staged in.</td>
<td>On primary execution host, and if successful, on all sister hosts allocated to job</td>
<td>See section 6.13.9.5.v, “execjob_begin: Event when Execution Host Receives Job”, on page 504.</td>
</tr>
<tr>
<td>execjob_prologue</td>
<td>An execjob_prologue hook is executed just before the first job process is started.</td>
<td>On primary execution host, and on all sister hosts where any job task is spawned or attached</td>
<td>See section 6.13.9.5.vi, “execjob_prologue: Event Just Before Execution of Top-level Job Process”, on page 505.</td>
</tr>
</tbody>
</table>
## Table 6-20: Event Type Objects

<table>
<thead>
<tr>
<th>Type</th>
<th>Trigger</th>
<th>Where Run</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>execjob_epilogue</td>
<td>An execjob_epilogue hook is executed after all of the job processes have terminated.</td>
<td>On all hosts allocated to the job</td>
<td>See section 6.13.9.5.viii, “execjob_epilogue: Event Just After Killing Job Tasks”, on page 508.</td>
</tr>
<tr>
<td>execjob_end</td>
<td>An execjob_end hook is executed on all hosts allocated to a job, at the end of all job processing</td>
<td>On all hosts allocated to the job</td>
<td>See section 6.13.9.5.vii, “execjob_end: Event After Job Cleanup”, on page 507.</td>
</tr>
<tr>
<td>execjob_preterm</td>
<td>An execjob_preterm hook is executed when the job receives a termination signal.</td>
<td>On all hosts allocated to the job</td>
<td>See section 6.13.9.5.ix, “execjob_preterm: Event Just Before Killing Job Tasks”, on page 510.</td>
</tr>
<tr>
<td>exechost_periodic</td>
<td>An exechost_periodic hook is executed at specified intervals</td>
<td>On all execution hosts in the complex</td>
<td>See section 6.13.9.5.x, “exechost_periodic: Periodic Events on All Execution Hosts”, on page 511.</td>
</tr>
<tr>
<td>resvsub</td>
<td>Triggered by pbs_rsub and the pbs_submitresv() API call. A resvsub hook is executed after all processing of pbs_rsub input, and just before a reservation is created.</td>
<td>At server</td>
<td>See section 6.13.9.5.xii, “resvsub: Event when Reservation is Created”, on page 512.</td>
</tr>
</tbody>
</table>
Chapter 6 Hooks

6.13.9.5.i queuejob: Event when Job is Queued

The event type for this event is pbs.QUEUEJOB.

A queuejob hook runs after all processing of qsub input, just before the job reaches the server, and before the job is queued, including when a job is peer queued to a server with a queuejob hook. The hook is triggered by qsub or the pbs_submit() API call. A queuejob hook is not triggered by requeuing a job (qrerun) or on node_fail_requeue, when a job is discarded by the MoM because the execution host went down.

A queuejob event hook can set any job attribute visible to qsub, can set the job’s comment, and can modify the resources requested by a job. See section 6.13.5, “Using Attributes and Resources in Hooks”, on page 456, for a complete listing of attributes and resources that this hook can modify.

A queuejob hook cannot modify the server or any queues.

In a queuejob event, the event’s job object members are as they would be if the job were to be successfully submitted.

A pbs.QUEUEJOB event has the following member, in addition to those listed in section 6.13.9.1, “Event Object Members for All Events”, on page 492:

```plaintext
pbs.event().job
```

A pbs.job object with the attributes and resources specified at submission for the job being queued. See section 6.13.12, “Job Objects”, on page 519. Some job attributes are read-only, as described in “Job Attributes” on page 375 of the PBS Professional Reference Guide, and cannot be reset.

A pbs.event().accept() terminates hook execution and allows the job to be queued, and any changes to job attributes or resources take effect.

A pbs.event().reject() terminates hook execution and causes the job not to be queued.

6.13.9.5.ii modifyjob: Event when Job is Altered

The type for this event is pbs.MODIFYJOB.

A modifyjob hook is executed after all processing of qalter input, and just before the job’s attributes are modified. The hook is triggered by qalter or the pbs_alterjob() API call.

Before the job runs, this hook can set any job attribute that can be changed via qalter, can set the job’s comment, and can set any resource requested by the job. While the job is running, the only job attributes and resources that the hook can set are those that can be changed via the qalter command. While the job runs, a modifyjob hook can change the job’s cput and walltime. See section 6.13.5, “Using Attributes and Resources in Hooks”, on page 456, for a complete listing of attributes and resources that this hook can modify.
See “qalter” on page 127 of the PBS Professional Reference Guide and “Job Attributes” on page 375 of the PBS Professional Reference Guide.

A modifyjob hook cannot change the server or any queues.

A job object’s attributes appear to a modifyjob hook as they would be after the job is modified, not before.

A modifyjob event hook shows the original job with all its attributes in pbs.event().job_o.

A pbs.MODIFYJOB event has the following members, in addition to those listed in section 6.13.9.1, “Event Object Members for All Events”, on page 492:

- pbs.event().job
  A pbs.job object containing the attributes and resources specified for the job being modified. See section 6.13.12, “Job Objects”, on page 519. Some job attributes are read-only, as described in “Job Attributes” on page 375 of the PBS Professional Reference Guide, and cannot be reset.

- pbs.event().job_o
  A pbs.job object representing the original job, before the job was modified via qalter. See section 6.13.12, “Job Objects”, on page 519.

A pbs.event().accept() terminates hook execution and allows the job to be altered, and any changes to job attributes or resources take effect.

A pbs.event().reject() terminates hook execution and causes the job not to be altered.

6.13.9.5.iii movejob: Event when Job is Moved

The type for this event is pbs.MOVEJOB.

The server runs its movejob hooks when any of the following happens:

- This server is the furnishing server when peer scheduling a job
- A job is moved from this server to another server via the qmove command
- A job is moved between two queues on this server

A movejob hook is executed after qmove arguments are processed, but before a job is moved from one queue to another. This hook is triggered by qmove and the pbs_movejob() API call. movejob hooks are not triggered by pbs_rsub -Wqmove=<job ID>.

The only job attribute that a movejob event hook can set is the job’s destination queue. A movejob hook can specify only local queues as the destination queue. Whether a job is submitted with a local queue or a remote queue as its destination, a movejob hook can change the destination to a local queue.

A movejob hook cannot modify the server or any queues.
Hooks

A job object’s attributes appear to a movejob hook as they would be after the event, not before it.

The hook shows the job’s current queue in the `pbs.event().src_queue` object member.

A `pbs.MOVEJOB` event has the following members, in addition to those listed in section 6.13.9.1, “Event Object Members for All Events”, on page 492:

- `pbs.event().job`
  - A `pbs.job` object representing the job being moved. See section 6.13.12, “Job Objects”, on page 519.
  - Note that `pbs.event().job.queue` refers to the destination queue, not the current queue.

- `pbs.event().src_queue`
  - The `pbs.queue` object representing the original queue where `pbs.event().job` came from.

A `pbs.event().accept()` terminates hook execution and allows the job to be moved, and any changes to job attributes or resources take effect.

A `pbs.event().reject()` terminates hook execution and causes the job not to be moved.

6.13.9.5.iv runjob: Event Before Job is Received by MoM

The event type is `pbs.RUNJOB`.

A `runjob` event occurs when one of the following happens:

- The administrator uses the `qrun` command
- The scheduler chooses to run a job and calls `pbs_runjob()`

A `runjob` hook is executed just before a job is sent to the execution host. It is triggered by `qrun` and the `pbs_runjob()` API call.

A `runjob` hook can modify a job both when the hook rejects the job and when the hook accepts the job.

On rejecting a job, a `runjob` hook can modify the following:

- The job’s `Hold_Types` attribute
- The job’s `Execution_Time` attribute
- The job’s `project` attribute
- The job’s `depend` attribute
- All of the job’s `Variable_List` attribute members
- All of the job’s `Resource_List` attribute members, except for `nodect`
On accepting a job, a runjob hook can modify the following:

- The job’s Error_Path attribute
- The job’s Output_Path attribute
- All of the job’s Variable_List attribute members
- The following Resource_List attribute members:
  - cput
  - exec_vnode
  - file
  - max_walltime
  - min_walltime
  - nice
  - pcput
  - pmem
  - pvmem
  - site
  - software
  - start_time
  - walltime

A runjob hook can modify a vnode only if the hook rejects the event, and the vnode is in the job’s exec_vnode attribute. For a vnode, the hook can modify only the state attribute. The only pre-execution event hook that can change this attribute is a runjob hook.

A runjob hook cannot modify the server or any queues.

For a runjob hook only, job object attributes appear as they would be before the event takes place.

A pbs.RUNJOB event has the following member, in addition to those listed in section 6.13.9.1, “Event Object Members for All Events”, on page 492:

```python
pbs.event().job
```

A pbs.job object representing the job being run. See section 6.13.12, “Job Objects”, on page 519.

A pbs.event().accept() terminates hook execution and allows the job to be sent to the execution host, and any changes to job attributes or resources take effect.

A pbs.event().reject() terminates hook execution and causes the job not to be sent to the execution host.
Chapter 6  

Hooks

6.13.9.5.v  execjob_begin: Event when Execution Host Receives Job

This event type is pbs.EXECJOB_BEGIN.

An execjob_begin hook executes on the primary execution host and then, if successful, executes on all the sister hosts allocated to the job. The hook executes when the host first receives the job, after any files or directories are staged in.

An execjob_begin hook can modify the job’s Execution_Time, Hold_TYPES, Variable_List, and resources_used attributes. It can also modify the attributes and available resources of the vnodes managed by the local MoM.

An execjob_begin hook can flag the job to be rerun.

A pbs.EXECJOB_BEGIN event has the following members and methods, in addition to those listed in section 6.13.9.1, “Event Object Members for All Events”, on page 492:

- pbs.event().job
  This is a pbs.job object representing the job that is about to run. See section 6.13.12, “Job Objects”, on page 519.

- pbs.event().vnode_list[]
  This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.13.9.2.i, “The Vnode List Member”, on page 493 for information about using pbs.event().vnode_list[].

- pbs.reboot()
  Reboots host. See section 6.13.9.4.i, “Method to Reboot Host”, on page 495.

A call to pbs.event().accept() means the job can proceed with execution, and any changes to job attributes, resources, or the vnode list take effect.

A call to pbs.event().reject(<message>) automatically causes the job to be killed and tells the server to requeue the job. In addition, any changes to job attributes, resources, or vnode list take effect.

- If the pbs.event().reject(<message>) call is made on a primary execution host, the following message appears in the MoM log at log event class 0x0100:
  “execjob_begin request rejected by <hook_name>”
  <message>

  The rejection message <message> also appears in the STDERR of the program (e.g. qrun) invoking pbs_runjob() API:

- If the pbs.event().reject(<message>) call is made on a sister host, the following mes-
Hooks

Chapter 6

Message appears in the MoM log at log event class 0x0100:
"execjob_begin request rejected by <hook_name>"

In addition, this message appears in mom_logs on the primary execution host:
"job_start_error: <hook errno> from node <hostname> could not JOIN_JOB successfully.

• If pbs_runjob() was invoked by the scheduler, the following job comment appears:
  "Not running; PBS Error: <message>"

If the execjob_begin hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, the job is automatically killed and the server requeues the job. All job changes, vnode changes, or requests for host reboot or scheduler cycle restarts do not take effect. In this case, one of the following messages appears in the MoM log at event class 0x0100:

  "execjob_begin hook <hook_name> encountered an exception, request rejected"

  "alarm call while running execjob_begin hook '<hook_name>', request rejected"

6.13.9.5.vi execjob_prologue: Event Just Before Execution of Top-level Job Process

This event type is pbs.EXECJOB_PROLOGUE.

An execjob_prologue hook runs on the primary execution host, and on each of the sister MoM hosts allocated to the job, if and when at least one of the job’s tasks is spawned or attached. On the primary execution host, an execjob_prologue hook executes just prior to executing the top-level shell or cmd process of the job. This is where the prologue executes. On a sister host, the hook executes just before the first task of the job on this host is spawned. It is not run for any additional spawned task on this host.

An execjob_prologue hook overrides a prologue. If an execjob_prologue hook exists and is enabled, MoM executes the hook. Otherwise, she executes the prologue.

An execjob_prologue hook can modify the job’s Execution_Time, Hold_Types, and resources_used attributes. It can also modify the attributes and available resources of the vnodes managed by the local MoM.

An execjob_prologue hook can flag the job to be rerun.

A pbs.EXECJOB_PROLOGUE event has the following members and methods, in addition to those listed in section 6.13.9.1, “Event Object Members for All Events”, on page 492:
Chapter 6  

Hooks

pbs.event().job
This is a pbs.job object representing the job that is about to run. See section 6.13.12, “Job Objects”, on page 519.

pbs.event().vnode_list[]
This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.13.9.2.i, “The Vnode List Member”, on page 493 for information about using pbs.event().vnode_list[].

pbs.reboot()
Reboots host. See section 6.13.9.4.i, “Method to Reboot Host”, on page 495.

A pbs.event().accept() allows the job to continue its normal execution, and any changes to job attributes, resources, or vnode list take effect.

A pbs.event().reject(<message>) causes the job to be killed, and the owning server to requeue the job. Any changes to job attributes, resources, or vnode list take effect.

• On the primary execution host, the following job-level mom_logs entries appear:
  “execjob_prologue request rejected by <hook_name>”
  <message>
• On a sister vnode, the following job-level mom_logs entries appear:
  “execjob_prologue request rejected by <hook_name>”
  <message>
• In addition, the following message appears in the STDERR of the program invoking the tm_attach() API, such as the pbs_attach command:
  “a hook has rejected the task manager request”

If the following setting is specified in the hook script, just before issuing a pbs.event().reject(), then the job is aborted (meaning the server deletes the job) instead of being requeued:

    pbs.event().job.delete()

If the user attribute of the execjob_prologue hook is set to pbsuser, the hook script executes under the context of the job owner (the value of the euser job attribute).
Hooks

If the execjob_prologue hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, the job is killed and the server requeues the job. All job changes, vnode changes, or requests for host reboot or scheduler cycle restarts, do not take effect. In addition, one of the following messages appears in the MoM log at event class 0x0100:

“execjob_prologue hook <hook_name> encountered an exception, request rejected”

“alarm call while running execjob_prologue hook '<hook_name>', request rejected”

The standard output and standard error of an execjob_prologue hook script are not connected to the standard output and standard error of the job.

6.13.9.5.vii execjob_end: Event After Job Cleanup

This event type is pbs.EXECJOB_END.

An execjob_end hook executes on all the hosts allocated to a job. The hook is executed after a job is cleaned up.

An execjob_end hook can modify the attributes and available resources of the vnodes managed by the local MoM.

An execjob_end hook cannot effectively modify the job’s Execution_Time and Hold_Types attributes. These changes will not be visible to the server, because the job is already cleaned up and reported.

A pbs.EXECJOB_END event has the following members and methods, in addition to those listed in section 6.13.9.1, “Event Object Members for All Events”, on page 492:

pbs.event().job
This is a pbs.job object representing the job that just ran. See section 6.13.12, “Job Objects”, on page 519.

pbs.event().vnode_list[]
This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.13.9.2.i, “The Vnode List Member”, on page 493 for information about using pbs.event().vnode_list[].

pbs.event().job.Exit_status
A Python int that holds the exit value of the top level shell of the job script. This value is valid only when the hook executes on a primary execution host.

pbs.reboot()
Reboots host. See section 6.13.9.4.i, “Method to Reboot Host”, on page 495.

A call to pbs.event().accept() ends the job, and any changes to job attributes, resources, or vnode list take effect.
A call to `pbs.event().reject(<message>)` also ends the job, and any changes to job attributes, resources, or vnode list also take effect.

A call to `pbs.event().reject(<message>)` on a primary execution host causes the following message to appear in the MoM log at log event class 0x0100:

```
"execjob_end request rejected by <hook_name>"
<message>
```

A call to `pbs.event().reject(<message>)` on a sister host causes the following message to appear in the MoM log at log event class 0x0100:

```
"execjob_end request rejected by <hook_name>"
<message>
```

If the `execjob_end` hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, the job terminates, and all job changes, vnode changes, requests for host reboot or scheduler cycle restarts, do not take effect. In addition, one of the following messages appear in the MoM logs at event class 0x0100:

```
"execjob_end hook <hook_name> encountered an exception, request rejected"
"alarm call while running execjob_end hook '<hook_name>', request rejected"
```

### 6.13.9.5.viii execjob_epilogue: Event Just After Killing Job Tasks

This event type is `pbs.EXECJOB_EPILOGUE`.

An `execjob_epilogue` hook executes on all the hosts allocated to the job. On a primary execution MoM host, the hook executes after all the job tasks/processes on the host have been killed, and basic CPU and memory resource usage information have been logged. This is where the epilogue executes. On a sister MoM, the hook executes after the sister MoM receives a request to kill the job and has signaled the job tasks to terminate.

An `execjob_epilogue` hook overrides an epilogue. If an `execjob_epilogue` hook exists and is enabled, MoM executes the hook. Otherwise, she executes the epilogue.

An `execjob_epilogue` hook can modify the job’s `Execution_Time`, `Hold_Types`, and `resources_used` attributes. It can also modify the attributes and available resources of the vnodes managed by the local MoM.

An `execjob_epilogue` hook can flag the job to be rerun.

A `pbs.EXECJOB_EPILOGUE` event has the following members and methods, in addition to those listed in section 6.13.9.1, “Event Object Members for All Events”, on page 492:

- `pbs.event().job`
  
  This is a `pbs.job` object representing the job that just finished. See section 6.13.12, “Job Objects”, on page 519.
pbs.event().vnode_list[]
This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the
vnodes that are assigned to this job. See section 6.13.9.2.i, "The Vnode List
Member", on page 493 for information about using pbs.event().vnode_list[].

pbs.event().job.Exit_status
A Python int that holds the exit value of the top level shell of the job script. This
value is valid only if the hook is executing on a primary execution host.

pbs.reboot()
Reboots host. See section 6.13.9.4.i, "Method to Reboot Host", on page 495.
A call to pbs.event().accept() continues the normal end-of-job processing, and any changes
to job attributes, resources, or vnode list take effect.
A call to pbs.event().reject() causes the job on the current vnode to exit, and the owning
server to completely delete the job. Any changes to job attributes, resources, or vnode list
take effect.

• On a primary execution host, a pbs.event().reject(<message>) causes the following
message to appear in the MoM log at log event class 0x0100:

"execjob_epilogue request rejected by <hook_name>"
"<message>

• On a sister host, a pbs.event().reject(<message>) causes the following message to
appear in the MoM log at log event class 0x0100:

"execjob_epilogue request rejected by <hook_name>"
"<message>

• If the following setting has been made prior to calling pbs.event().reject(), the owning
server requeues the job:

pbs.event().job.rerun()
If the user attribute of the execjob_prologue hook is set to pbsuser, the hook script executes
under the context of the job owner (the value of the euser job attribute).

If the execjob_epilogue hook script encounters an unexpected error causing an unhandled
exception, or if the script terminates due to a hook alarm, this causes the job on the current
vnode to exit, and the owning server to completely delete the job. All job changes, vnode
changes, requests for host reboot or scheduler cycle restarts, do not take effect. In addition,
one of the following messages appears in the MoM log at event class 0x0100:

"execjob_epilogue hook <hook_name> encountered an exception, request
rejected"
"alarm call while running execjob_epilogue hook '<hook_name>', request
rejected"
Chapter 6  

Hooks

The standard output and standard error of an execjob_epilogue hook script are not connected to the standard output and standard error of the job.

6.13.9.5.ix  execjob_preterm: Event Just Before Killing Job Tasks

This event type is pbs.EXECJOB_PRETERM.

An execjob_preterm hook executes on all the hosts allocated to a job. This hook runs only when a qdel has been issued. It does not run for any other job termination. For example, it does not run on a qrerun or when a job goes over its limit. On the primary execution host, the hook executes when the job receives a signal from the server for the job to terminate. On a sister host, this hook executes when the sister receives a request from the primary execution host to terminate the job, just before the sister signals the task on this host to terminate.

An execjob_preterm hook can modify the job’s Execution_Time, Hold_Types, and resources_used attributes. It can also modify the attributes and available resources of the vnodes managed by the local MoM.

A pbs.EXECJOB_PRETERM event has the following members and methods, in addition to those listed in section 6.13.9.1, “Event Object Members for All Events”, on page 492:

- **pbs.event().job**
  - This is a pbs.job object representing the job that is about to run (or be killed).
  - See section 6.13.12, “Job Objects”, on page 519.

- **pbs.event().vnode_list**
  - This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.13.9.2.i, “The Vnode List Member”, on page 493 for information about using pbs.event().vnode_list[].

- **pbs.reboot()**
  - Reboots host. See section 6.13.9.4.i, “Method to Reboot Host”, on page 495.

A pbs.event().accept() call allows job cancellation or deletion to happen, and any changes to job attributes, resources, or vnode list take effect.

A pbs.event().reject() call causes the job instance on a vnode to continue running, because the terminate signal is not delivered to the job. Any changes to job attributes, resources, or vnode list take effect.

- On the primary execution host, a pbs.event().reject(<message>) causes the following to appear in the STDERR of the program (qdel) invoking the pbs_deljob() API:
  - “hook rejected request”
- The following message appears in the MoM log at log event class 0x0100:
  - “execjob_preterm request rejected by <hook_name>”
  - <message>
- On a sister host, a pbs.event().reject(<message>) causes the following message to
Hooks

appear in the MoM log at log event class 0x0100:
  “execjob_preterm request rejected by <hook_name>”
  <message>
If the _user_ attribute of the _execjob_prologue_ hook is set to _pbsuser_, the hook script executes under the context of the job owner (the value of the _euser_ job attribute).

If the _execjob_preterm_ hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, the job continues to run, and all job changes, vnode changes, requests for host reboot or scheduler cycle restarts, do _not_ take effect. In addition, one of the following messages appears in the MoM log at event class 0x0100:
  “execjob_preterm hook <hook_name> encountered an exception, request rejected”
  “alarm call while running execjob_preterm hook '<hook_name>', request rejected”

6.13.9.5.x  _exechost_periodic_: Periodic Events on All Execution Hosts

This event type is _pbs.EXECHOST_PERIODIC_.

The _exechost_periodic_ hook runs periodically on all the hosts in the complex where a _pbs_mom_ is running.

The interval between calls to _exechost_periodic_ hooks is specified in the _freq_ hook attribute. See _section 6.11.6, “Setting Hook Frequency”, on page 451_.

An _exechost_periodic_ hook can modify the attributes and resources of the vnodes managed by the local MoM.

A _pbs.EXECHOST_PERIODIC_ event has the following members and methods, in addition to those listed in _section 6.13.9.1, “Event Object Members for All Events”, on page 492_:

  _pbs.event().vnode_list[]_
  This is a dictionary of _pbs.vnode_ objects, keyed by vnode name, listing the vnodes that are managed by the MoM where the hook runs. See _section 6.13.9.2.i, “The Vnode List Member”, on page 493_ for information about using _pbs.event().vnode_list[]_.

  To get the vnode managed by the local MoM, use the _pbs.get_local_nodename()_ function to return the local natural vnode name where this hook is executing, and then use _pbs.event().vnode_list[<local natural vnode name>]_.

  _pbs.reboot()_
  Reboots host. See _section 6.13.9.4.i, “Method to Reboot Host”, on page 495_.

PBS Professional 12.2 Administrator’s Guide  511
Chapter 6

Hooks

A call to `pbs.event().accept()` or `pbs.event().reject(<message>)` causes any changes made to vnodes to take effect.

A call to `pbs.event().reject(<message>)` causes the following messages to appear in the MoM log:

```
"exechost_periodic" request rejected by hook <hook_name>"
<message>
```

The periodic hook continues to be periodically called whether or not there are errors in hook script execution or a call to the `pbs.event().reject()` action. To stop the hook from being called, either disable it or delete it:

```
#qmgr -c "s h <periodic hook> enabled=f
#qmgr -c "d h <periodic hook>
```

If the `exechost_periodic` hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, all vnode changes, requests for host reboot or scheduler cycle restarts, do **not** take effect. In addition, one of the following messages appears in the MoM log at event class 0x0100:

```
"exechost_periodic hook <hook_name> encountered an exception, request rejected"
"alarm call while running exechost_periodic hook '<hook_name>', request rejected"
```

### 6.13.9.5.xi Caveats for Periodic Event Hooks

The `order` attribute is ignored for `exechost_periodic` hooks. It does **not** guarantee the execution order of a list of periodic hooks.

### 6.13.9.5.xii resvsub: Event when Reservation is Created

The type for this event is `pbs.RESVSUB`.

A `resvsub` hook is executed after all processing of `pbs_rsub` input, and just before a reservation is created. The hook is triggered by `pbs_rsub` and the `pbs_submitresv()` API call.

The only time that a reservation can be modified is during its creation. A `resvsub` event hook can set any settable reservation attribute and any resource that can be specified via `pbs_rsub`. See section 6.13.5.3.iii, “Using Reservation Attributes in Hooks”, on page 465 for a complete list of the reservation attributes that this hook can read and set.

A reservation object’s attributes appear to a `resvsub` hook as they would be after the event, not before it.
A `pbs.RESVSUB` event has the following member, in addition to those listed in section 6.13.9.1, “Event Object Members for All Events”, on page 492:

```plaintext
pbs.event().resv
```

A `pbs.resv` object containing the attributes and resources specified for the reservation being requested. See section 6.13.15, “Reservation Objects”, on page 526.

A `pbs.event().accept()` terminates hook execution and allows creation of the reservation, and any changes to reservation resources take effect.

A `pbs.event().reject()` terminates hook execution and causes the reservation not to be created.
6.13.9.6 Event Object Examples

Example 6-14: Inside a hook script, create a PBS event object:
   
   e = pbs.event()

Example 6-15: Get the event type:
   
   type = e.type

Example 6-16: Get the user who requested the event action:
   
   who = e.requestor

Example 6-17: Get the host where the request came from:
   
   host = e.requestor_host

Example 6-18: The event type is pbs.QUEUEJOB. Get the number of CPUs requested for the job being queued:
   
   j = e.job
   res = j.Resource_List["ncpus"]

Example 6-19: Reset the number of CPUs requested by the job:
   
   j.Resource_List["ncpus"] = 1

Example 6-20: The event type is pbs.MOVEJOB. Get the request parameters:
   
   j = e.job
   q = j.queue

Example 6-21: Accept an event request:
   
   e.accept()

Example 6-22: Reject an event request:
   
   e.reject("Can't set interactive attribute")

Example 6-23: Put a job into a wait state and requeue the job in 3600 seconds (1 hour):
   
   import time
   ...
   j.Execution_Time = time.time() + 3600

Example 6-24: Put a hold on a job:
   
   j = pbs.event().job
   j.Hold_Types = pbs.hold_types("u")
   j.Hold_Types = pbs.hold_types("uo")
   j.Hold_Types += pbs.hold_types("s")
or

```python
j.Hold_Types = pbs.hold_types("<hold_list>")
```

Example 6-25: Release a hold on a job:

```python
j.Hold_Types = pbs.hold_types("un")
j.Hold_Types = pbs.hold_types("sp")
j.Hold_Types = pbs.hold_types("o")
or
j.Hold_Types = pbs.hold_types("<hold_list>")
```

### 6.13.10 Server Objects

This object represents a PBS server. This object can either represent the local server, or be just a coding construct, not representing an actual server. If it represents the local server, you can read but cannot set its attributes. If it is just a coding construct that does not represent an actual server, you can set its attributes. You cannot alter the PBS server. If this server object represents the PBS server, it is the server at which the triggering event is taking place, and at which the hook is executing. The only PBS server available to hooks is the local server.

```python
s = pbs.server(['<name>'])
```

Creates an instance of a PBS server object. If `<name>` is not specified, the object represents the default server.

You can use `pbs.server()` to retrieve server, queue, job, vnode, and reservation information, and pass it to a hook script. You cannot set attributes for objects that are retrieved through the server via `pbs.server()`.

#### 6.13.10.1 Server Object Members

An instance `s` of `pbs.server` has the following members:

```python
s.name
```

The server hostname.

Example: `myhost.mydomain.com`

This member is read-only.

Python type: `str`

```python
s.<attribute name>
```

The PBS server attribute named `<attribute name>`. Server attributes are listed in "Server Attributes" on page 316 of the PBS Professional Reference Guide.
6.13.10.1.i Setting Server Object Members

If the server object does not represent the PBS server, you can set, but not unset, server object members. If a server object does represent the PBS server, you cannot set values for object members. To set the value for the server attribute named <attribute name> to <attribute value>:

s.<attribute name> = <attribute value>

6.13.10.1.ii Examples of Using Server Object Attributes

Example 6-26: Get all the attributes and their values for the current PBS complex:

s = pbs.server()

Example 6-27: Get server name:

name = s.name

Example 6-28: Get the value of the server attribute pbs_license_min:

min = s.pbs_license_min

6.13.10.2 Server Object Methods

s.job('<id>')

Returns a pbs.job object for the job with ID <id>, residing on the local server. Returns None if the job with ID <id> does not exist at the server. See section 6.13.12, “Job Objects”, on page 519.

s.jobs()

Returns a Python iterator that iterates over a list of pbs.job objects residing on the local server. Returns an empty iterator if no jobs exist on the local server. The iterator is described in section 6.13.18, “Other Global Creation Methods”, on page 534.

Example:

for j in s.jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "found job \%s\n" % (j.id))

s.queue(<queue_name>)

Returns a queue object representing the queue named <queue name> that is managed by the local server. See section 6.13.11, “Queue Objects”, on page 518.

A value of None is returned if the queue named <queue_name> does not exist at the local server.
s.queues()  
Returns a Python iterator that iterates over a list of queue objects managed by
the local server. Returns an empty iterator if no queues exist at the local
server. The iterator is described in section 6.13.18, “Other Global Creation
Methods”, on page 534.

s.resv( <reservation ID> )  
Returns a pbs.resv object for <reservation ID> on the local server. Returns
None if <reservation ID> does not exist.

s.resvs()  
Returns a Python iterator that iterates over a list of pbs.resv objects residing on
the local server. Returns an empty iterator if no reservations exist at the local
server. The iterator is described in section 6.13.18, “Other Global Creation
Methods”, on page 534.

s.scheduler_restart_cycle()  
This directs the current PBS server to tell the scheduler to restart its scheduling
cycle.

A hook with its user attribute set to pbsuser cannot successfully invoke
pbs.scheduler_restart_cycle(), unless the hook’s executing user is a PBS Man-
ger or Operator. If this is attempted, the scheduler is not restarted, and the fol-
lowing message appears at log event class 0x0100 in the MoM logs:

"<node_host_name>;Not allowed to update vnodes or to request
scheduler_restart_cycle, if run as a non-manager/operator
user"

This method can be invoked from all hooks.

s.vnode( <vnode name> )  
Returns a pbs.vnode object representing the vnode with name <vnode name>
that is managed by the current server.

Returns None if <vnode name> does not exist.

s.vnodes()  
Returns a list of pbs.vnode objects managed by current server.

Returns an empty iterator if no vnodes exist at the local server.

Example:

for vn in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, “found vn %s” % (vn.name))

---

Hooks  
Chapter 6
6.13.11 Queue Objects

This object represents a PBS queue. This object can either represent an actual PBS queue, or be just a coding construct, not representing an actual queue. If it is just a coding construct, you can set its attributes. If it represents an actual queue, you can read but cannot set its attributes. You cannot set the attributes of any actual queue in any hook.

`pbs.queue`

To get information about a particular queue with name `<name>`, you must go through the associated server. Use:

```python
q = pbs.server().queue("<name>")
```

To get a list of queues from the server:

```python
pbs.server().queues()
```

6.13.11.1 Queue Object Attributes

An instance `q` of `pbs.queue` has the following members representing attributes:

- `q.name`
  The queue name.
  This member is read-only.
  Python type: `str`

- `q.<attribute name>`
  The queue attribute named `<attribute name>`. Queue attributes are listed in “Queue Attributes” on page 354 of the PBS Professional Reference Guide.

Example 6-29: Get the queue object representing the queue `workq`, and its Priority value:

```python
q = s.queue("workq")
prio = q.Priority
```

6.13.11.1.i Setting Queue Object Attributes

You can set, but not unset, queue object attributes for queue objects that don’t represent an actual queue. To set the value of a queue object attribute named `<attribute name>`:

```python
q.<attribute name> = <attribute value>
```
6.13.11.2 Queue Object Methods

q.job( <job ID> )
Returns a pbs.job object representing PBS job with ID <job ID>. This job must be residing on the queue represented by q. Returns None if the job with the specified job ID does not exist, or if the job is not in the queue represented by q. See section 6.13.12, “Job Objects”, on page 519.

q.jobs()
Returns a Python iterator that iterates over a list of pbs.job objects representing the jobs on the queue q. Returns an empty iterator if no jobs exist on the queue q. The iterator is described in section 6.13.18, “Other Global Creation Methods”, on page 534.

Example:
for j in pbs.server().queue(“workq”).jobs():
    pbs.logmsg(pbs.LOG_DEBUG, “found job %s” % (j.id))

6.13.12 Job Objects

A job object represents a PBS job:

pbs.job

You can retrieve the job object either through an associated event or through the server. The job object represents one of the following, depending on how it is retrieved:

• The PBS job associated with the event that triggers the hook. To get the job associated with the current event, go through the event that triggered the hook:
  pbs.event().job
  A call to pbs.event().job can return only the job associated with the current event.
  When you get a job using pbs.event().job, the hook can read and set the job attributes and resources listed in section 6.3, “Job Attributes Readable & Settable via Events”, on page 458.

• A job at the server at which the hook is executing. To get a particular job with ID <id>, go through the server:
  pbs.server().job("<job ID>")
  When you get a job using pbs.server().job( <job ID> ), the hook can read all job attributes, but can set none.

To get a list of jobs at the server:

pbs.server().jobs()
Chapter 6

Hooks

All job objects have the same members and methods. Each hook can read or set different attributes and resources. We describe what each type of hook can do in section 6.13.9, “Event Objects”, on page 492.

If you use a hook to make a change to a job, that change is visible to all PBS daemons.

6.13.12.1 Job Object Members

A pbs.job object has the following member:

```python
class pbs.job:
    def __init__(self):
        self.id = None
```

- **pbs.job.id**
  - Represents the PBS job ID.
  - Read-only.
  - Python type: `str`

6.13.12.1.i Job Object Attributes

A pbs.job object has a member to represent each job attribute. Each member is spelled exactly like the corresponding attribute. For event-retrieved jobs, the following members can be set in various hooks:

- **pbs.job.Account_Name**
  - Python type: `str`

- **pbs.job.block**
  - Python type: `int`

- **pbs.job.Checkpoint**
  - Python type: `pbs.checkpoint`

- **pbs.job.depend**
  - Python type: `pbs.depend`

- **pbs.job.Error_Path**
  - Python type: `str`

- **pbs.job.Execution_Time**
  - Represents the time when the current job is eligible to run. Example of setting:
    ```python
    pbs.job.Execution_Time = time.mktime([<YY>, <MM>, <DD>, <HH>,
                                           <MM>, <SS>, <WEEKDAY>, <YEARDAY>, <ISDST>])
    ```
  - Python type: `int`

For example, the following sets a job's Execution_Time to: March 1, 2012 at 09:00 am:

```python
pbs.job.Execution_Time = time.mktime([2012, 3, 1, 12, 9, 0, -1, -1, -1])
```

Python type: `int`
pbs.job.Hold_Types
    Python type: pbs.hold_types

pbs.job.Job_Name
    Python type: str

pbs.job.Join_Path
    Python type: pbs.join_path

pbs.job.Keep_Files
    Python type: pbs.keep_files

pbs.job.Mail_Points
    Python type: pbs.mail_points

pbs.job.Mail_Users
    Python type: pbs.email_list

pbs.job.Output_Path
    Python type: str

pbs.job.Priority
    Python type: int

pbs.job.Queue
    Python type: pbs.queue

pbs.job.Rerunnable
    Python type: bool

pbs.job.Resource_List[<resource name>]
    Represents the job’s Resource_List attribute.
    Python type: dictionary: Resource_List["<resource name>"]=<value> where
    <resource name> is any built-in or custom resource

pbs.job.resources_used[<resource name>]
    Represents job’s resources_used attribute, which lists the resources used by
    the job.
    If an execution hook modifies specific resources used by a job, MoM refrains
    from updating those values.
    Under Linux/UNIX, the resources that MoM does not modify if they’ve been set
    in a hook are cput, walltime, mem, vmem, ncpus, and cpupercent.
    Under Windows, the resources that MoM does not modify if they’ve been set in
    a hook are cput, walltime, mem, and ncpus.
Chapter 6

Hooks

If you use a non-execjob_end execution hook to set a value for resources_used, the new value for resources_used appears in the accounting logs.

If you use an execjob_end execution hook to set a value for resources_used, it has no effect, because MoM has already sent the final values for resources_used to the server.

Python type: dictionary: resources_used["<resource name>"]=<value> where <resource name> is any built-in or custom resource

pbs.job.run_count
Execution hooks must run with user = pbsadmin to reduce the value of this member. Execution hooks running with user = pbsuser cannot reduce the value of this member.

Python type: int

pbs.job.sandbox
Python type: str

pbs.job.stagein
pbs.job.stageout
Python type: pbs.staging_list

pbs.job.umask
Python type: int

pbs.job.User_List
Python type: pbs.user_list

pbs.job.Variable_List[<variable>]
Holds the job’s environment variables. Example of setting a value:

pbs.job.Variable_List[<variable>] = <value>

Python type: dictionary: Variable_List["<variable name>"]=<value>

The remaining members cannot be set in hooks. Job object member types are listed in “Job Attributes” on page 375 of the PBS Professional Reference Guide.

6.13.12.1.ii Setting Job Attributes

You can set any settable attributes for the job, and any resource used by the job, as follows:

pbs.event().job.<attribute> = <value>

pbs.event().job.Resource_List["<resource name>"] = <value>

See section 6.15.7, “Rules for Working With Attributes”, on page 553.
6.13.12.1.iii Examples of Using Job Object Attributes

Get the job's Priority value:

```python
demo = pbs.job.Priority
```

Reset the Priority value of job `j`:

```python
demo.Priority = 5
```

Get the job's PBS_O_WORKDIR environment variable:

```python
demo.Variable_List['PBS_O_WORKDIR']
```

### 6.13.12.2 Job Object Methods for Execution Hooks

All job objects that are associated with events have the following methods, which are available only in execution hooks (the event name is prefixed with "execjob_"):

- **`pbs.job.is_checkpointed()`**
  - Returns a Python `bool` value which is `True` if the job was checkpointed under the control of the PBS MoM.
  - For example, you could use this in an `execjob_epilogue` hook, where the hook writer directs the job to be requeued if the job was checkpointed under the control of PBS:
    ```python
    # cat epi.py
    import pbs
    if pbs.event().job.is_checkpointed():
        pbs.event().job.rerun()
        pbs.event().reject("job to be requeued")
    # qmgr -c "create hook epi event=execjob_epilogue"
    # qmgr -c "import hook epi application/x-python default epi.py"
    ```

- **`pbs.job.in_ms_mom()`**
  - Returns a Python `bool` value. Returns `True` if this job object is running on the primary execution host.

- **`pbs.job.delete()`**
  - When this method is used in an execution hook, the job is flagged at the server for deletion after its processes have terminated and any epilogue or `execjob_epilogue` hook has run.
  - When this method is used in a non-execution hook script, it raises a Python "`NotImplementedError`" exception.
If the `pbs.job.delete()` method is used in an `execjob_end` hook, it has no effect, because in this case the server has already performed end-of-job processing before the execution hook runs.

The `pbs.job.delete()` method overrides the `pbs.job.rerun()` method. If both are used, `pbs.job.delete()` takes precedence.

`pbs.job.rerun()`
When this method is used in an execution hook, the job is flagged at the server for requeueing after its processes have terminated and any epilogue or `execjob_epilogue` hook has run.

When this method is used in a non-execution hook script, it raises a Python “`NotImplementedError`” exception.

If the `pbs.job.rerun()` method is used in an `execjob_end` hook, it has no effect, because in this case the server has already performed end-of-job processing before the execution hook runs.

The `pbs.job.delete()` method overrides the `pbs.job.rerun()` method. If both are used, `pbs.job.delete()` takes precedence.

### 6.13.13 exec_vnode Object

The `exec_vnode` object represents the job’s `exec_vnode` attribute:

```python
pbs.exec_vnode
```

The `exec_vnode` object contains a list of chunks where the job runs. Each chunk contains the vnode name and the assigned resources. See section 6.13.14, “Chunk Objects”, on page 526.

### 6.13.13.1 exec_vnode Object Attributes

A `pbs.exec_vnode` object has the following member:

```python
pbs.exec_vnode.chunks[]
```

List of `pbs.vchunk` objects. These objects represent the chunks assigned to a job.
Example:

```python
j = pbs.event().job
pbs.logmsg(pbs.LOG_DEBUG, "job %s exec_vnode = %s" % (j.id, j.exec_vnode))

evc = j.exec_vnode.chunks
for c in evc:
    pbs.logmsg(pbs.LOG_DEBUG, "c.vnode_name=%s " % (c.vnode_name))
    for r in c.chunk_resources.keys():
        pbs.logmsg(pbs.LOG_DEBUG, "c.chunk_resources[%s]=%s" % (r, c.chunk_resources[r]))
```

Sample output:

```
10:16:53;0006;Server@jobim;Hook;Server@jobim;job 153.jobim
exec_vnode = (jobim[2]:ncpus=2:mem=10240kb)+
    (jobim[3]:ncpus=2:mem=10240kb) + (jobim[3]:ncpus=2:mem=2048kb)
10:16:53;0006;Server@jobim;Hook;Server@jobim; jobim[2]
    c.vnode_name=jobim[2]
    c.chunk_resources[ncpus]=2
    c.chunk_resources[mem]=10240kb
10:16:53;0006;Server@jobim;Hook;Server@jobim; jobim[1]
    c.vnode_name=jobim[1]
    c.chunk_resources[ncpus]=2
    c.chunk_resources[mem]=10240kb
10:16:53;0006;Server@jobim;Hook;Server@jobim; jobim[3]
    c.vnode_name=jobim[3]
    c.chunk_resources[ncpus]=2
    c.chunk_resources[mem]=2048kb
```

### 6.13.13.2 Restrictions on `exec_vnode` Objects

A job’s `exec_vnode` attribute is read-only. You cannot set its value, and you cannot build an `exec_vnode` object using `pbs.vchunk` objects.
Chapter 6  

Hooks

6.13.14  Chunk Objects

The pbs.vchunk object represents a chunk specification. It is used in a job’s exec_vnode attribute or select statement:

```bash
pbs.vchunk
```

6.13.14.1  Chunk Object Attributes

A pbs.vchunk object has the following members representing attributes:

- **pbs.vchunk.vnode_name**
  - Name of the vnode from which the chunk is taken.
  - Python type: `str`

- **pbs.vchunk.chunk_resources**
  - Resources assigned to the chunk.
  - Python dictionary containing resource `name=value` pairs.
  - Format: `chunk_resources['<resource name>'] = <resource val>` where `<resource name>` is any custom or built-in resource. See Chapter 5, "Resources", on page 297.

6.13.14.2  Chunk Object Methods

```python
vc.chunk_resources.keys()
```

- The list of `<resource name>` keys of `chunk_resources`. This list makes it convenient to list all the values of `chunk_resources`. Example:

```python
for r in vc.chunk_resources.keys():
    pbs.logmsg(pbs.LOG_DEBUG, "c.chunk_resources[%s]=%s" % (r, vc.chunk_resources[r]))
```

6.13.15  Reservation Objects

This represents a PBS reservation. If the reservation is associated with the triggering event, you can read and set reservation attributes and resources. See section 6.13.5.3.iii, “Using Reservation Attributes in Hooks”, on page 465 for a complete list of the reservation attributes and resources that can be set in the resvsub hook. If the reservation is retrieved through the server, and is not associated with the triggering event, you can read all its attributes and resources, but set none.
If you are working with the reservation being created using `pbs_rsub`, you must use `pbs.event().resv`. The server cannot return information about the reservation, because it has not yet been created.

```python
pbs.resv
```

In order to retrieve information about the reservation associated with the triggering action, you must use a reference to the reservation object represented by:

```python
pbs.event().resv
```

To get a copy of a particular reservation, use:

```python
pbs.server().resv("<reservation name>")
```

To get a list of the reservations at a server:

```python
pbs.server().resvs()
```

### 6.13.15.1 Reservation Object Attributes

A `pbs.resv` object has the following members:

```python
pbs.resv.resvid
```

The reservation ID.

Example: “R221.myhost”.

This member is read-only.

Python type: `str`

```python
pbs.resv.<attribute name>
```

The reservation attribute named `<attribute name>`. To see a list of which reservation attributes can be read and set by each hook, see Table 6-5, “Reservation Attributes Readable & Settable via Events,” on page 465.

### 6.13.15.1.i Setting Reservation Object Attribute Values

You can set, but not unset, reservation object attributes. We list which reservation attributes can be set or read in section 6-5, “Reservation Attributes Readable & Settable via Events”, on page 465. To set a reservation object attribute:

```python
pbs.resv.<attribute name> = <attribute value>
```
6.13.15.1.ii Examples of Using Reservation Object Attributes

Example 6-30: Get the reservation’s owner:

```python
owner = pbs.resv.Reserve_Owner
```

Example 6-31: Reset the reservation’s name:

```python
pbs.resv.Reserve_Name = “Resv2008”
```

6.13.16 Vnode Objects

This represents a PBS vnode.

`pbs.vnode`

The way in which you retrieve a vnode controls what you can do with the vnode. If a vnode is retrieved through an event, and is managed by the same MoM where the event hook runs, you can set the vnode attributes and resources listed in Table 6-4, “Vnode Attributes Readable & Settable via Events,” on page 463. However, if a vnode is not retrieved through an event, or is not managed by the same MoM where the hook runs, you can read all vnode attributes and resources, but set none.

- To retrieve the list of vnodes associated with an execution event or a periodic event:
  ```python
  pbs.event().vnode_list[]
  ```
- To retrieve a specific vnode that is associated with an execution or periodic event, use the list of vnodes associated with the event, and specify the vnode name:
  ```python
  pbs.event().vnode_list[“<vnode name>”]
  ```
- To retrieve the vnodes associated with a pre-execution event, get the job’s `exec_vnode` attribute:
  ```python
  pbs.event().job.exec_vnode
  ```
  This gives you a list of chunks in `pbs.event().job.exec_vnode.chunks[]`. If `ec` is a chunk, `ec.vnode_name` contains the name of the vnode.
- To retrieve the server’s list of vnodes:
  ```python
  pbs.server().vnodes()
  ```
- To retrieve a named vnode through the server:
  ```python
  pbs.server().vnode(“<vnode name>”)
  ```

6.13.16.1 Vnode Object Attributes

A `pbs.vnode` object has a member representing each attribute. Table 6-4, “Vnode Attributes Readable & Settable via Events,” on page 463 lists which vnode attributes can be set by each hook.
When this list is retrieved through an execution event, it is associated with a job, and only
vnodes assigned to the job have attributes, resources_available, resourcesassigned.ncpus, and resourcesassigned.mem filled in; on other vnodes, only
pbs.vnode().name is available. You can set any settable attributes for the job’s vnodes, and
any resources_available at the job’s vnodes, as follows:

```
pbs.event().vnode_list[<vnode name>].<attribute> = <value>
pbs.event().vnode_list[<vnode name>].resources_available["<resource name>"] = value
```

### 6.13.16.1 Vnode Attribute Restrictions

- The only vnode attribute that can be changed by a pre-execution hook is the state
  attribute
- The only pre-execution hook that can change the vnode state attribute is the runjob hook
- Execution and periodic hooks can change all settable vnode attributes

### 6.13.17 Global Attribute Creation Methods

These methods can be used to create attribute objects for any object, although some make
sense only with certain objects. Some attributes are used with more than one object. For
example, the email_list() creation method can be used for both reservation and job
Mail_Users attributes. These methods are globally available.

```python
pbs.acl("[+\-]<entity>][,...]")
```

Creates an object representing a PBS ACL, using the given string parameter.

Instantiation of these objects requires a formatted input string.

```python
pbs.checkpoint("<checkpoint_string>")
```

where `<checkpoint_string>` must be one of "n", "s", "c", "c=mmm", "w", or
"w=mmm"

Creates an object representing the job Checkpoint attribute, using the given
string.

Instantiation of these objects requires a formatted input string.

```python
pbs.depend("<depend_string>")
```

where `<depend_string>` must be of format "<type>:<jobid>,<jobid>...", or
"on:<count>".

where `<type>` is one of "after", "afterok", "afterany", "afternotok", "before",
"beforeok", "beforeany", and "beforenotok".

Creates a PBS dependency specification object representing the job depend
attribute, using the given `<depend_string>`.
Chapter 6  

Hooks

Usage:

```python
pbs.event().job.depend = pbs.depend("<depend_string>")
```

Instantiation of these objects requires a formatted input string.

```python
pbs.duration("[[hours:]minutes:]seconds[.milliseconds]")
```

Creates a time specification `duration` instance, returning the equivalent number of seconds from the given time string. Represents an interval or elapsed time in number of seconds. Duration objects can be specified using either a time or an integer. See the "`pbs.duration(<integer>)`" creation method.

```python
pbs.duration(<integer>)
```

Creates an integer `duration` instance using the specified number of seconds.

A `pbs.duration` instance can be operated on by any of the Python `int` functions. When performing arithmetic operations on a `pbs.duration` type, ensure the resulting value is a `pbs.duration()` type, before assigning to a job member that expects such a type.

Example:

```python
pbs.event().job.Resource_List["cput"] = pbs.duration(300 + d1) # safe
```

The following will not work, since Python evaluates from left to right, and returns result as the type at left (`int`):

```python
d1 = pbs.duration(30)
pbs.event().job.Resource_List["cput"] = 300 + d1
```

```python
pbs.email_list("<email_address1>, <email address2>..."")
```

Creates an object representing a mail list.

Instantiation of these objects requires a formatted input string.

```python
pbs.exec_host("host/N[*C][+...]")
```

Create an object representing the `exec_host` job attribute, using the given host and resource specification.

Instantiation of these objects requires a formatted input string.

```python
pbs.exec_vnode("<vchunk>[+<vchunk> ...]"
```

`<vchunk>` is `(<vnodename:ncpus=N:mem=M>)`

Creates an object representing the `exec_vnode` job attribute, using the given vnode and resource specification. When the `qrun -h` command is used, or when the scheduler runs a job, the `pbs.job.exec_vnode` object contains the vnode specification for the job.

Instantiation of these objects requires a formatted input string.
Example:

```python
pbs.exec_vnode("(vnodeA:ncpus=N:mem=X)+(nodeB:ncpus=P:mem=Y+nodeC:mem=Z)")
```

This object is managed and accessed via the `str()` or `repr()` functions. Example:

```python
import pbs

server = pbs.Server()
job = server.j('ID000155510-620200765-w000000000-000000000-1')
ev = job.exec_vnode

print(str(ev))
```

```
"(vnodeA:ncpus=2:mem=200m)+(vnodeB:ncpus=5:mem=1g)"
```

`pbs.group_list("<group_name>[[@<host>]|<group_name>[[@<host>]|...]]")`

Creates an object representing a PBS group list.

To use a group list object:

```python
pbs.job.group_list = pbs.group_list(...)  # path to a group list
```

Instantiation of these objects requires a formatted input string.

`pbs.hold_types("<hold_type_str>")`

where `<hold_type_str>` is one of "u", "o", "s", or "n".

Creates an object representing the `Hold_Types` job attribute.

Instantiation of these objects requires a formatted input string.

`pbs.job_sort_formula("<formula string>")`

where `<formula string>` is a string containing a math formula. See section 4.8.20, "Using a Formula for Computing Job Execution Priority", on page 182.

Creates an object representing the `job_sort_formula` server attribute.

Instantiation of these objects requires a formatted input string.

`pbs.join_path({"oe","eo","n"})`

Creates an object representing the `Join_Path` job attribute.

Instantiation of these objects requires a formatted input string.

`pbs.keep_files("<keep_files_str>")`

where `<keep_files_str>` is one of "o", "e", "oe", "eo".

Creates an object representing the `Keep_Files` job attribute.

Instantiation of these objects requires a formatted input string.


Instantiates an object representing a `license_count` attribute.

Instantiation of these objects requires a formatted input string.

`pbs.mail_points("<mail_points_str>")`

where `<mail_points_str>` is "$a", "$b", and/or "$e", or "$n".  

Example:
Chapter 6

Hooks

Creates an object representing a Mail_Points attribute.
Instantiation of these objects requires a formatted input string.

```
pbs.node_group_key("<resource>")
```

Creates an object representing the resource to be used for node grouping, using the specified resource.

```
pbs.path_list("<path>@<host>[,<path>@<host> ...]")
```

Creates an object representing a PBS pathname list.

To use a path list object:

```
pbs.job.Shell_Path_List = pbs.path_list(...)
```

Instantiation of these objects requires a formatted input string.

```
pbs.place("[arrangement]:[sharing]:[group]")
```

- **arrangement** can be "pack", "scatter", "free", "vscatter"
- **sharing** can be "shared", "excl", "exchost"
- **group** can be of the form "group=<resource>"

[arrangement], [sharing], and [group] can be given in any order or combination.

 Creates a place object representing the job’s place specification.

Instantiation of these objects requires a formatted input string.

Example:

```
pl = pbs.place("pack:excl")
s = repr(pl)          (or s = `pl`)  
letter = pl[0]        (assigns 'p' to letter)
s = s + ":group=host" (append to string)
pl = pbs.place(s)     (update original pl)
```

```
pbs.range("<start>-<stop>:<step>")
```

Creates a PBS object representing a range of values.

Example:

```
pbs.range("1-30:3")
```

Instantiation of these objects requires a formatted input string.

```
pbs.route_destinations("<queue_spec>[,<queue_spec>,...]")
```

where <queue_spec> is queue_name[@server_host[:port]]

Creates an object that represents a route_destinations routing queue attribute.

Instantiation of these objects requires a formatted input string.
pbs.select("[N:]res=val[,res=val] [+[N:]res=val[,res=val] ... ]")
Creates a select object representing the job’s select specification.
Instantiation of these objects requires a formatted input string.
Example:

```python
sel = pbs.select("2:ncpus=1:mem=5gb+3:ncpus=2:mem=5gb")
s = repr(sel)  # (or s = `sel`)
letter = s[3]  # (assigns 'c' to letter)
s = s + "+5:scratch=10gb"  # (append to string)
sel = pbs.select(s)  # (reset the value of sel)
```

pbs.size(<integer>)
Creates a PBS size object using the given integer value, storing the value as the number of bytes. Size objects can be specified using either an integer or a string. See the "pbs.size("<integer><suffix>")" creation method.

pbs.size("<integer><suffix>")
Creates a PBS size object out of the given string specification. The suffix must be a multiplier defined in the table shown in “Size” on page 403 of the PBS Professional Reference Guide. The size of a word is the word size on the execution host. Size objects can be specified using either an integer or a string.
To operate on pbs.size instances, use the “+” and “-” operators.
To compare pbs.size instances, use the “==”, “! =”, “>”, “<”, “>=”, and “<=” operators.
Example: the sizes are normalized to the smaller of the 2 suffixes. In this case, “10gb” becomes “10240mb” and is added to “10mb”:

```python
sz = pbs.size("10gb")
sz = sz + 10mb
10250mb
```
Example: the following returns True because sz is greater than 100 bytes:

```python
if sz > 100:
gt100 = True
```

pbs.staging_list("<filespec>[,<filespec>,...]")
where <filespec> is <execution_path>@<storage_host>:<storage_path>
Creates an object representing a job file staging parameters list.
To use a staging list object:

```python
pbs.job.stagein = pbs.staging_list(...)```

PBS Professional 12.2 Administrator’s Guide 533
Instantiation of these objects requires a formatted input string.

```python
```

Instantiates an object representing a `state_count` attribute.

Instantiation of these objects requires a formatted input string.

```python
pbs.user_list("<user>[@<host>][,<user>@<host>...]")
```

Creates an object representing a PBS user list.

To use a user list object:

```python
pbs.job.User_List = pbs.user_list(...)
```

Instantiation of these objects requires a formatted input string.

### 6.13.18 Other Global Creation Methods

The following methods are available in any hook.

```python
pbs.args("<args>")
```

where `<args>` are space-separated arguments to a command such as `qsub` or `qdel`.

Creates an object representing the arguments to the command.

Example:

```python
pbs.args("-Wsuppress_email=N -r y")
```

Instantiation of these objects requires a formatted input string.

```python
pbs.get_local_nodename()
```

This returns a Python `str` whose value is the name of the local natural vnode.

If you want to refer to the vnode object representing the current host, you can pass this vnode name as the key to `pbs.event().vnode_list[]`. For example:

```python
Vn = pbs.event().vnode_list[pbs.get_local_nodename()]
```

```python
pbs.logjobmsg(job ID, message)
```

where `job ID` must be an existing or previously existing job ID and where `message` is an arbitrary string.

This puts a custom string in the PBS Server log. The `tracejob` command can be used to print out the job-related messages logged by a hook script.

Messages are logged at log event class `pbs.LOG_DEBUG`.

```python
pbs.logmsg(log event class, message)
```

where `message` is an arbitrary string, and where `log event class` can be one of the message log event class constants:
Hooks

pbs.LOG_WARNING
pbs.LOG_ERROR
pbs.LOG_DEBUG

See “Message Log Event Class Objects” on page 487.

This puts a custom string in the PBS Server log.

Example:

```python
for j in pbs.server().jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "found job %s" % (j.id))
```

pbs.software("<software info string>")

Creates an object representing a site-dependent software resource.

Instantiation of these objects requires a formatted input string.

pbs.version("<pbs version string>")

Creates an object representing the PBS version string.

Instantiation of these objects requires a formatted input string.
6.14 Hook Examples

Example 6-32: This PBS hook script rejects jobs which do not specify walltime.

Script RequireWalltime.py:

```python
import pbs
import sys

try:
e = pbs.event()
j = e.job
if j.Resource_List["walltime"] == None :
    e.reject("Job has no walltime requested")
except SystemExit:
    pass
except pbs.UnsetResourceNameError:
    e.reject("Job has no walltime requested")
```

Create hook and import script:

```
qmgr -c 'create hook RequireWalltime event="queuejob"
qmgr -c 'import hook RequireWalltime application/x-python default Require-Walltime.py'
```

Example 6-33: This PBS hook script rejects jobs with CPU requests that are not multiples of 8:

Script Multiple8.py:

```python
import pbs
import sys

e = pbs.event()
j = e.job
```
mult_limit = 8

if j.Resource_List["ncpus"] != None:
    try:
        e = pbs.event()
        j = e.job
        R = j.Resource_List["ncpus"] % mult_limit
        if R != 0:
            e.reject("Ncpus resource is not a multiple of %s." % (mult_limit,))
    except SystemExit:
        pass
    except (pbs.UnsetResourceNameError, TypeError):
        e.reject("Bad ncpus resource value.")

else:
    R = pbs.event().job.Resource_List
    sel = repr(R["select"])
    tot_ncpus = 0
    for chunk in sel.split("+"):
        nchunks = 1
        for c in chunk.split(":"):
            kv = c.split("=")
            if len(kv) == 1:
                nchunks = kv[0]
            elif len(kv) == 2:
                if kv[0] == "ncpus":
                    tot_ncpus += (int(nchunks) * int(kv[1]))
    try:
        mod = tot_ncpus % mult_limit
        if mod != 0:
            e.reject("Ncpus resource is not a multiple of %s." % \ (mult_limit,))
    except SystemExit:
        pass
except(pbs.UnsetResourceNameError,TypeError):
    e.reject("Bad Ncpus resource value.")

Create hook and import script:

```
qmgr -c 'create hook Multiple8 event="queuejob"'
qmgr -c 'import hook Multiple8 application/x-python default Multiple8.py'
```

Example 6-34: If a user asks for `-l ncpus=8:ppn=24`, change `ncpus` to 24

Script `ChangeNcpus.py`:

```
import pbs
import sys

try:
    e = pbs.event()
    j = e.job
    j.Resource_List["ncpus"] = max(j.Resource_List["ncpus"],
        j.Resource_List["ppn"])
except SystemExit:
    pass

except(pbs.UnsetResourceNameError, pbs.BadResourceValError):
    e.reject("Failed to reset ncpus value")

Create hook and import script:

```
qmgr -c 'create hook ChangeNcpus event="queuejob"
qmgr -c 'import hook ChangeNcpus application/x-python default ChangeNc-
pus.py'
```

Example 6-35: Calculate `cph` (`cph == total ncpus * walltime` (in hours)) and set a custom `cph` job resource to the value. (You must create the `cph` resource before using it.)

Script `CustCPH.py`:

```
import pbs
```

538
R = pbs.event().job.Resource_List
sel = repr(R["select"])  
tot_ncpus = 0
for chunk in sel.split("+"):
    nchunks = 1
    for c in chunk.split(":"):
        kv = c.split("=")
        if len(kv) == 1:
            nchunks = kv[0]
        elif len(kv) == 2:
            if kv[0] == "ncpus":
                tot_ncpus += (int(nchunks) * int(kv[1]))
    R["cph"] = tot_ncpus * R["walltime"]

Create hook and import script:

```
qmgr -c 'create hook CustCPH event="queuejob"'
qmgr -c 'import hook CustCPH application/x-python default CustCPH.py'
```

Example 6-36: This PBS hook script puts a job in a particular queue (e.g. "interQ") if the job was submitted interactively (i.e. qsub -I)

Script IQueue.py:

```python
# get the pbs module
import pbs
import sys

try:
    # Get the hook event information and parameters
    # This will be for the 'queuejob' event type.
    e = pbs.event()

    # Get the information for the job being queued
    j = e.job
    if j.interactive:
        # Get the "interQ" queue object
        q = pbs.server().queue("interQ")
```
# Reset the job's destination queue
# parameter for this event
j.queue = q
# accept the event
e.accept()
except SystemExit:
pass
except:
e.reject("Failed to route job to queue interQ")

Create hook and import script:

```bash
qmgr -c 'create hook IQueue event="queuejob"'
qmgr -c 'import hook IQueue application/x-python default IQueue.py'
```

Example 6-37: Prevent users from using `qalter` to change their jobs in any way. Allow only administrators to change jobs.

Script `NoAlter.py`, on Windows, in a domain:

```python
import os
import pbs

e = pbs.event()
j = e.job
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))
isadmin=0
admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "Administrator"]

if who in admin_ulist:
isadmin=1
else:
    cmd = "net user " + who + " /domain"
    admin_glist = ['Administrators', 'Domain Admins', 'Enterprise
    Admins']
    for line in os.popen(cmd).readlines():
        if line.find("Group") >= 0:
            for li in line.split("*")):
Hooks

Chapter 6

```python
if li.strip() in admin_glist:
    isadmin=1
break
if e.type == pbs.MODIFYJOB and not isadmin:
    e.reject("Normal users are not allowed to modify their jobs")

Script NoAlter.py, on Linux/UNIX:

import pbs
e = pbs.event()
j = e.job
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))
admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "root"]
if who not in admin_ulist:
    e.reject("Normal users are not allowed to modify their jobs")

Create hook and import script:

```
qmgr -c 'create hook NoAlter event="modifyjob"
qmgr -c 'import hook NoAlter application/x-python default NoAlter.py'
```

Example 6-38: Restrict the ability to submit a reservation to PBS administrators only.

Script NoSub.py on Windows:

```python
import pbs
import os
e = pbs.event()
r = e.resv
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))
isadmin=0

admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "Administrator"]
if who in admin_ulist:
    isadmin=1
```
else:
    cmd = "net user " + who + "/domain"
admin_glist = ['Administrators', 'Domain Admins', 'Enterprise Admins']
for line in os.popen(cmd).readlines():
    if line.find("Group") >= 0:
        for li in line.split("*"):
            if li.strip() in admin_glist:
                isadmin=1
                break
if e.type == pbs.RESVSUB and not isadmin:
    e.reject("Only admins allowed to create reservations!")

Script NoSub.py on Linux/Unix:

```python
import pbs
import os
e = pbs.event()
r = e.resv
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))

admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "root"]

if e.type == pbs.RESVSUB and who not in admin_ulist:
    e.reject("Only admins allowed to create reservations!")

Create hook and import script:

```
cmgr -c 'create hook NoSub event="resvsub"'
cmgr -c 'import hook NoSub application/x-python default NoSub.py'
```

Example 6-39: Reject jobs that request a specific queue, e.g. workq2, that do not request memory.

Script queuespec.py:

```python

```
import pbs
import sys
try:
e = pbs.event()
j = e.job
if j.queue.name == "workq2" and not j.Resource_List["mem"]:
    e.reject("workq2 requires job to have mem specification")
except SystemExit:
    pass
except:
e.reject("%s hook failed with %s. Please contact Admin" % (e.hook_name, sys.exc_info()[2]))

Create hook, import script:

    qmgr -c 'create hook queuespec event="modifyjob"
    qmgr -c 'import hook queuespec application/x-python default queuespec.py'

Example 6-40: Set job’s project based on queue where job is submitted. The following is a snippet of a queuejob hook:

import pbs
e = pbs.event()
If e.job.queue == None:
    # user did not specify a queue to submit to, so use default
target_qname = pbs.server().default_queue
else:
target_qname = e.job.queue.name
If (target_qname == "large") or (target_qname == "medium"):
e.job.project = "some_large_medium_project"
Example 6-41: Speed up throughput of interactive jobs. Use a queuejob hook that determines whether a job entering the system is an interactive job. If so, it directs the job to the high priority queue specified in 'high_priority_queue', and tells the server to restart the scheduling cycle. You must first define a "high" queue as follows:

```bash
qmgr -c "create queue high queue_type=e,Priority=150"
qenable high
qstart high
```

The default priority for an express queue is 150. If you do not want interactive jobs to go into an express queue, set the priority of the queue named "high" to a value greater than ordinary queues but lower than the value for an express queue. See section 4.8.17, "Express Queues", on page 169.

Instantiate the hook as follows:

```bash
qmgr -c "create hook rapid_inter event=queuejob"
qmgr -c "import hook rapid_inter application/x-python default rapid_inter.py"
```

Hook script:

```python
import pbs

high_priority_queue="high"

e = pbs.event()
if e.job.interactive:
    high = pbs.server().queue(high_priority_queue)
    if high != None:
        e.job.queue = high
        pbs.logmsg(pbs.LOG_DEBUG, "quick start interactive job")
        pbs.server().scheduler_restart_cycle()
```
Example 6-42: Monitor the load average on the local host using a periodic hook. Offline or free the vnode representing the host depending on the CPU load. You can modify values for ideal_load and max_load. Your hook does the following:

If the system’s CPU load average rises above max_load, the state of the vnode corresponding to the current host is set to offline. This prevents the scheduler from scheduling jobs on this vnode.

If the system’s CPU load average falls below ideal_load, the state of the vnode representing the current host is set to free. This allows the scheduler to schedule jobs on this vnode.

To instantiate this hook, specify the following:

```
qmgr -c "create hook load_balance event=exechost_periodic,freq=10"
qmgr -c "import hook load_balance application/x-python default load_balance.py"
```

Hook script:

```python
import pbs
import os
import re

ideal_load=1.5
max_load=2.0

# get_la: returns a list of load averages within the past 1-minute, 5-minute, 15-minutes range.
def get_la():
    line=os.popen("uptime").read()
    r = re.search(r'load average: \S+, \S+, \S+$', line).groups()
    return map(float, r)

local_node = pbs.get_local_nodename()

vnl = pbs.event().vnode_list
current_state = pbs.server().vnode(local_node).state
mla = get_la()[0]
if (mla >= max_load) and ((current_state == pbs.ND_OFFLINE) == 0):
    vnl[local_node].state = pbs.ND_OFFLINE
    vnl[local_node].comment = "offlined node as it is heavily loaded"
```

elif (mla < ideal_load) and ((current_state == pbs.ND_OFFLINE) != 0):
    vnl[local_node].state = pbs.ND_FREE
    vnl[local_node].comment = None

Example 6-43: A hook script that periodically updates the values of a set of custom resources for the vnode where the current MoM runs.

The current set includes two size types, which are scratch and home

Prerequisites:
1. Define the following custom resources in server's resourcedef file, and restart pbs_server.
   % cat PBS_HOME/server_priv/resourcedef
   scratch type=size flag=nh
   home type=size flag=nh

2. Add the new resources to the "resources:" line in the sched_config file and restart pbs_sched:
   % cat PBS_HOME/sched_priv/sched_config resources
   ncpus, mem, arch, [...], scratch, home

3. Install this hook as follows:
   qmgr -c "create hook mom_dyn_res event=exechost_periodic,freq=30"
   qmgr -c "import hook mom_dyn_res application/x-python default
           mom_dyn_res.py"

The mom_dyn_res.py script:
# NOTE:
# Update the dyn_res[] array below to include any other custom resources
# to be included in the updates. Ensure that each resource added has an
# entry in the server's resourcedef file and scheduler's sched_config
# file.

import pbs
import os
import sys

# get_filesystem_avail_unprivileged: returns available size in kbytes
# (in pbs.size type) to unprivileged users, of the filesystem where
# 'dirname' resides.
def get_filesystem_avail_unprivileged( dirname):
    o = os.statvfs(dirname)
    return pbs.size( "%skb" % ((o.f_bsize * o.f_bavail) / 1024) )

# get_filesystem_avail_privileged: returns available size in kbytes
# (in pbs.size type) to privileged users, of the filesystem where 'dirname'
# resides.

def get_filesystem_avail_privileged( dirname):
    o = os.statvfs(dirname)
    return pbs.size( "%skb" % ((o.f_bsize * o.f_bfree) / 1024) )

try:
    # Define here the custom resources as key, and the function and its
    # argument for obtaining the value of the custom resource:
    #    Format: dyn_res[<resource_name>] = [<function_name>,
    #        <function_argument>]
    # So "<function_name>(<function_argument>)" is called to return the
    # value for custom <resource_name>.
    dyn_res = {}
    dyn_res["scratch"] = [get_filesystem_avail_unprivileged, "/tmp"]
    dyn_res["home"] = [get_filesystem_avail_unprivileged, "/home"]

    vnl = pbs.event().vnode_list
    local_node = pbs.get_local_nodename()

    for k in dyn_res.keys():
        vnl[local_node].resources_available[k] =
        dyn_res[k][0](dyn_res[k][1])

except SystemExit:
    pass

except:
    e = pbs.event()
6.15 Advice and Caveats

6.15.1 Rules for Hook Access and Behavior

The following are rules and recommendations for writing hooks:

- Please read section 6.5, “Updating Hooks and Custom Resources”, on page 419.
- Use only the documented interfaces. Hooks which access PBS information or modify
  PBS in any way except through these interfaces are erroneous and unsupported.
- Do not attempt to manipulate the hook stored by PBS.
- Don’t delete attributes.
- Don’t change environment variables set by PBS. See “Environment Variables” on page
  229 of the PBS Professional Reference Guide for a list of these environment variables.
- Do not try to access the following (a well-written, portable hook will not depend on any
  of the following information):
  - Server configuration information: qmgr, resourcedef and pbs.conf
  - Scheduling information: qmgr, sched_config, fairshare, dedicated, holidays
  - Vnode information: qmgr, pbsnodes
- Do not write hooks that depend on the behavior of other hooks.
- Do not make assumptions about the value of PATH; use “import sys” and “modify
  sys.path”
- Do not make assumptions about the value of the current working directory.
- For information about umask, see “qalter” on page 127 of the PBS Professional Refer-
  Attributes” on page 375 of the PBS Professional Reference Guide.
- Do not depend on order of execution of unrelated hooks. For example, do not depend on
  one job submission’s queuejob hooks running entirely before another job submission’s
  queuejob hooks. It is not guaranteed that all of one job’s hooks will finish before another
  job’s hooks start.
- The Resource_List attribute is listed as a Python dictionary. These dictionaries support a
  restricted set of operations. They can reference values by index. Other features, such as
has_key(), are not available.

- Hooks which execute PBS commands are erroneous and unsupported. The behavior of executing PBS commands inside a hook is undefined (and is likely to cause the hook to hang).

### 6.15.2 Referencing the Object You Want

In order to read or operate on an object, you must retrieve it through the event or through the server. You can operate only on objects retrieved through `pbs.event()`. If you retrieve the object through `pbs.server()`, the object is read-only. See section 6.13.5.1, “How Object is Retrieved Affects Access”, on page 456.

#### 6.15.2.1 Referencing Jobs

In order to get read-only information about a particular job with ID `<id>`, use `pbs.server().job('<id>')`. This returns a read-only copy of the job. To get the job object representing the job associated with the current event, on which you can operate, use `pbs.event().job`.

#### 6.15.2.2 Referencing Reservations

In order to get information about a reservation being created in a `resvsub` event, use `pbs.event().resv`. `pbs.server()` cannot return information about the reservation, because the reservation has not yet been created.

#### 6.15.2.3 Referencing Destination Queues

Hooks have access only to the local server. Hooks can allow a job submission to a remote server, but they cannot specify a remote server. See section 6.15.10, “Local Server Only”, on page 564. Hooks can specify the destination queue at a local server for a `queuejob` or `move-job` event, whether the original destination queue was at the local server or a remote server.

To specify a destination queue at the local server:

```plaintext
pbs.event().job.queue = pbs.server().queue("<local_queue>")
```

Do not specify a queue at a remote server in a hook script.

### 6.15.3 Adding and Deleting Vnodes

- Whenever you add a new MoM to the PBS server, all existing execution and periodic
hooks are automatically sent to the new MoM.

- When you delete the last vnode managed by a given MoM, PBS deletes all the execution and periodic hooks on that MoM.
- If you delete a vnode and then re-create it, the parent MoM may have leftover execution or periodic hooks that should have been deleted. If this happens, re-install the PBS MoM, then add the MoM to the server.

### 6.15.4 Printing And Logging Messages

Hooks can log a custom string in the Server’s log, at message log event class \texttt{pbs.LOG_DEBUG (0x0004)}. This is done using the \texttt{pbs.logjobmsg(job ID, message)} facility. See “\texttt{pbs.logjobmsg(job ID, message)}” on page 534

Hooks can specify a message for use when the corresponding action is rejected. This message is printed to \texttt{stderr} by the command that triggered the event, and is printed in the daemon’s log. This is done using the \texttt{pbs.logmsg()} function. See “\texttt{pbs.logmsg(log event class, message)}” on page 534 for information on how to specify a rejection message.

Hooks cannot directly print to \texttt{stdout} or \texttt{stderr}, or read from \texttt{stdin}. See section \texttt{6.15.9.2, “Avoid Hook File I/O”, on page 560}, and section \texttt{6.17.8.8, “Hooks Attempting I/O”, on page 589}.

### 6.15.5 Capturing Return Code

To capture an application’s return code, you capture the return code in Python and then return it from the hook. You can use the Python \texttt{subprocess} module. Here is an example snippet:

```python
import sys
if "<path to subprocess module>" not in sys.path:
    sys.path.append("<path to subprocess module>")
import subprocess

try:
    retcode = subprocess.call("mycommand myarg", shell=True)
except OSError:
    retcode = -1

return retcode
```
6.15.6 Recommended Hook Script Structure

6.15.6.1 Catching Exceptions

Your hook script should catch all exceptions except for `SystemExit`.

It is helpful if it displays a useful error message in the `stderr` of the command triggering the hook. The error message should show the type of the error and should describe the error.

Here is the recommended script structure:

```python
import pbs
Import sys

try:
    ...

except SystemExit:
    pass

except:
    pbs.event().job.rerun()
    e.reject("%s hook failed with %s. Please contact Admin" % (e.hook_name, sys.exc_info()[:2]))
```
6.15.6.1.i Example of Catching Exceptions

This example shows how a coding error in the hook is caught with the except statement, and an appropriate error message is generated. In line 7, the statement \( k = \frac{5}{0} \) generates a divide-by-zero error. The hook script is designed to reject interactive jobs that are submitted to queue “nointer”.

```python
import pbs
import sys

try:
    batchq = "nointer"
    e = pbs.event()
    j = e.job
    k = 5/0
    if j.queue and j.queue.name == batchq and j.interactive:
        e.reject("Can't submit an interactive job in 'nointer' queue")
    except SystemExit:
        pass
    except:
        e.reject("%s hook failed with %s. Please contact Admin" % (e.hook_name, sys.exc_info()[:2]))

The hook is triggered:

% qsub job.scr
qsub: c1 hook failed with (type 'exceptions.ZeroDivisionError'), ZeroDivisionError('integer division or modulo by zero',)). Please contact Admin
6.15.6.2 Checking for Parameter Validity

To make hook scripts more robust, check first for the validity of the event parameters before using them, by comparing against None:

```python
if pbs.event().job != None:
    if pbs.event().job_o != None:
        if pbs.event().src_queue != None:
            if pbs.event().resv != None:
                if pbs.event().vnode != None:
                    if pbs.event().aoe != None:
```

6.15.6.2.i Example of Checking Validity

```
% cat t2245.ty
import pbs
e = pbs.event()
if e.type == pbs.QUEUEJOB & & (e.job == None):
    e.reject("Event Job parameter is unset!")
else if e.type == pbs.MODIFYJOB & & ((e.job == None) || (e.job_o == None)):
    e.reject("Event Job or Job_o parameter is unset!")
else if e.type == pbs.RESVSUB & & (e.resv == None):
    e.reject("Event Resv parameter is unset!")
else if e.type == pbs.RUNJOB & & (e.job == None):
    e.reject("Event Job parameter is unset!")
```

6.15.6.3 Minimize Unnecessary Steps

To speed up your hooks, move any steps to where they are used the fewest times possible. For example, if you retrieve several pieces of information about a job, but only use them if one of them fits a certain criterion, put the bulk of the information-retrieval steps in the section where you do the work on the job.

6.15.7 Rules for Working With Attributes

6.15.7.1 Using Job Object Time Attributes

For the attributes Execution_Time, ctime, etime, mtime, qtime, and stime, the pbs.job object expects or shows the number of seconds since Epoch. The only one of these that can be set is Execution_Time.
If you wish to set the value for **Execution_Time** using the `[CCYY]MMDDhhmm[.ss]` format, or to see the value of any of the time attributes in the ASCII time format, then load the Python `time` module and use the functions `time.mktime([CCYY, MM, DD, hh, mm, ss, -1, -1, -1])` and `time.ctime()`.

Example:

```python
import time
pbs.job.Execution_Time = time.mktime([07, 11, 28, 14, 10, 15, -1, -1, -1])
time.ctime(pbs.job.Execution_Time)
'Wed Nov 28 14:10:15 2007'
```

### 6.15.7.2 Converting `walltime` to Seconds

If you want to see a job’s `walltime` in seconds:

```python
int(pbs.event().job.Resource_List["walltime"])
```

For example:

```python
pbs.logmsg(pbs.LOG_DEBUG, "walltime=%d" %
(int(pbs.event().job.Resource_List["walltime"])))
```

If `walltime` is "00:30:15", this results in the following:

```
walltime=1815
```

### 6.15.7.3 Special Characters in `Variable_List` Job Attribute

When special characters are used in `Variable_List` job attributes, they must be escaped. For this attribute, special characters are comma (,), single quote (‘), double quote (“), and backslash (\). PBS requires each of these to be escaped with a backslash. However, Python requires that double quotes and backslashes also be escaped with a backslash. If the special character inside a string is a single quote, you must enclose the string in double quotes. If the special character inside the string is a double quote, you must enclose the string in single quotes. The following rules show how to use special characters in a `Variable_List` attribute when writing a Python script:

<table>
<thead>
<tr>
<th>Character</th>
<th>Example Value</th>
<th>How Value is Represented in Python Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>, (comma)</td>
<td>a,b</td>
<td>“a,b” or ‘a\,b’</td>
</tr>
</tbody>
</table>

| 554 | PBS Professional 12.2 Administrator’s Guide |
For example, if the path is:

```
"\Documents and Settings\pbstest\bin:\windows\system32"
```

This is how the path shows up in a script:

```python
pbs.job.Variable_List["PATH"] = "\Documents and Settings\pbstest\bin:\windows\system32"
```

### 6.15.7.4 Special Characters in `string_array` Attributes

For an attribute whose type is `string_array` and whose value contains one or more commas ("",""), the whole string must be enclosed in single quotes, outside of its double quotes. For example:

In `PBS_HOME/server_priv/resourcedef`:

```plaintext
test_string_array type=string_array
```

Value of `test_string_array` is:

```plaintext
glad, elated
```

In hook script:

```python
pbs.job.Resource_List["test_string_array"] = '"glad, elated"
```

### 6.15.7.5 Objects Requiring a Formatted Input String

Instantiation of some of the interface objects requires a formatted input string. You can operate on these objects only as if they are strings.

Instantiation of these objects requires a formatted input string. A `repr()` of the object's instance returns its full string representation. This representation can be manipulated using the built-in methods for Python `str`.

---

**Table 6-21: How to Use Special Characters in Python Scripts**

<table>
<thead>
<tr>
<th>Character</th>
<th>Example Value</th>
<th>How Value is Represented in Python Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>' (single quote)</td>
<td>c’d</td>
<td>“c\’d”</td>
</tr>
<tr>
<td>&quot; (double quote)</td>
<td>f&quot;g&quot;h</td>
<td>‘f\&quot;g\&quot;h’</td>
</tr>
<tr>
<td>\ (backslash)</td>
<td>\home\dir\files</td>
<td>“\home\dir\files” or “\home\dir\files’”</td>
</tr>
</tbody>
</table>

For example, if the path is:

```
"\Documents and Settings\pbstest\bin:\windows\system32"
```

This is how the path shows up in a script:

```python
pbs.job.Variable_List[“PATH”] = “\Documents and Settings\pbstest\bin:\windows\system32”
```
6.15.7.6 Format of Reservation Object Time Attributes

For the attributes `reserve_start`, `reserve_end`, and `ctime`, the `pbs.resv` object expects and shows the number of seconds since Epoch. The `ctime` attribute cannot be set.

If you want to set the value for `reserve_start` or `reserve_end` using the "/[CCYY/MM/DDhh:mm:ss]" format, or see the value for any time attribute displayed in the ASCII time format, load the Python time module and use the functions `time.mktime([CCYY, MM, DD, hh, mm, ss, -1, -1, -1])` and `time.ctime()`.

Example:

```python
import time
pbs.job.reserve_start = time.mktime([07, 11, 28, 14, 10, 15, -1, -1, -1])
time.ctime(pbs.job.reserve_start)
'Wed Nov 28 14:10:15 2007'
```

If `reserve_duration` is unset or set to `None`, the reservation’s duration is taken from the `wall` resource attribute associated with the reservation request. If `reserve_duration` and `wall` are both specified, meaning not set to `None`, `reserve_duration` will take precedence.

6.15.7.7 Rules for Setting and Unsetting Job Object Attributes and Resources

If a job object is retrieved through the server, all hooks can read all job attributes and resources, but set none. If a job object is retrieved through an event, the hook can read or set certain job attributes and resources, listed in Table 6-3, “Job Attributes Readable & Settable via Events,” on page 458. For the cases where you can set a job attribute or resource:

- If the member is of type `str`, `int`, `bool`, `long`, or `float`, you can set it this way:
  ```python
  pbs.job.<attribute name> = <attribute value>
  ```

- If the member is of any other type, you must use the corresponding attribute creation method. For example:
  ```python
  pbs.job.Hold_Types = pbs.hold_types("uo")
  ```

  For creation methods, see section 6.13.17, “Global Attribute Creation Methods”, on page 529.

- When you unset an attribute or resource, it takes its default value. To unset an attribute or resource, set `<attribute value>` to `None`:
  ```python
  pbs.job.<attribute name> = None
  ```

- You cannot unset a job attribute that has a creation method.
6.15.7.8 Comparing Member to Constant Value or Constant Object

If an object member is of type `str`, `int`, `bool`, `long`, or `float`, you can compare it directly to a constant value of the same type.

Constant objects are integers, so if an object member is not one of those types, you must get the integer value of the member. First `str()` the member and then `int()` the member. For example, the `pbs.queue_type` member must be treated this way to compare it to the `pbs.QTYPE` constants:

```python
q = pbs.server().queue("workq")
qtype = int(str(q.queue_type))
if qtype == pbs.QTYPE_EXECUTION:
    pbs.logmsg(pbs.LOG_DEBUG, "queue %s is an execution type queue" % (q.name))
else:
    pbs.logmsg(pbs.LOG_DEBUG, "queue %s has unknown type" % (q.name))
```

6.15.7.9 Use Fast Operations

The examples we provide here could be faster. Instead of using `==`, you can use the bitwise ampersand operator (`&`).
6.15.8 Avoiding Interference with Normal Operation

6.15.8.1 Treating SystemExit as a Normal Occurrence

Both `pbs.event().accept()` and `pbs.event().reject()` terminate hook execution by throwing a `SystemExit` exception. A `try...except` clause without arguments will catch all exceptions. If hook content appears in a `try except` clause, add the following to treat `SystemExit` as a normal occurrence:

```python
except SystemExit:
    pass
```

Here is an example of an `except` clause that will catch `SystemExit`:

```python
try:
    ....
except:
    ....
```

In the above case, we need to add the `except SystemExit`, so that it will look like this:

```python
try:
    ....
except SystemExit:
    pass
except:
    ....
```

If the existing code has a specified exception, we don't need to add "except SystemExit;", since this hook script is only catching one particular exception and will not match `SystemExit`. For example:

```python
try:
    ...
except pbs.BadAttributeValueError:
    ...
```
6.15.8.2 Allowing the Server and Scheduler to Modify Jobs

The Server and Scheduler both use the `qalter` command during normal operation to modify jobs. Therefore, if you have a `modifyjob` hook script, make sure you do not interfere with `qalter` commands issued by the Server or Scheduler. Catch these cases by starting the hook with an `if` clause that accepts modification of jobs by PBS:

```python
e = pbs.event()
if e.requestor in ["PBS_Server", "Scheduler"]:
e.accept()
```

6.15.8.3 Using Sleep in a Windows Hook Script

Under Windows, the PBS server cannot interrupt a hook script executing the Python `time.sleep()`. The server needs to be able to interrupt the script if the script reaches its time-out. In order to be able to interrupt the script, create a sleep that incrementally sleeps for 1 second. The server can then interrupt the hook script in between the sleeps. For example:

```python
import time
def mysleep(sec):
    for i in range(sec):
        time.sleep(1)
mymy_sleep(30)  # <-- pseudo sleep for 30 seconds
```

6.15.8.4 Staying Within the Scheduler’s Alarm Time

Consider setting hook `alarm` values so that they do not unduly delay the scheduler. The scheduler will wait for a hook to finish executing. The scheduler’s cycle time has a default value of 20 minutes, and is specified in the scheduler’s `sched_cycle_length` attribute.

6.15.9 Avoiding Problems

6.15.9.1 Do Not Delete Jobs

Update your execution event hooks according to the procedure in section 6.5.3.1, “Caveats for Hooks that Use Custom Resources”, on page 420.
6.15.9.2 Avoid Hook File I/O

When the PBS server is running, stdout, stderr, and stdin are closed. A hook script attempting I/O will get an exception. To avoid this, redirect input and output to a file. See section 6.17.8.8, “Hooks Attempting I/O”, on page 589.

6.15.9.3 Avoid Contacting Bad Host

Be careful not to specify a bad host in <job ID> in pbs.event().job.depend. If it references a non-existent or heavily loaded PBS server, the current PBS server could hang for a few minutes as it tries to contact the bad host. For example:

```python
pbs.event().job.depend = pbs.depend("after:23.bad_host")
```

The PBS server could hang while trying to contact "bad_host".

6.15.9.4 Avoid os._exit() Python Function

Do not use the os._exit() Python function. It will cause the PBS server to exit.

6.15.9.5 Avoid Attempting to Log Message Using Bad Job ID

If the pbs.logjobmsg() method is passed a bad job ID, it raises a Python ValueError.

6.15.9.6 Avoid Taking Up Lots of Memory

Certain function calls in PBS Python hooks are expensive to use in terms of memory. If they are called repeatedly in loops, they can use up a lot of memory, potentially causing the server to hang or crash. For example, the following is expensive since each iterative call to `pbs.server().vnodes()` causes internal allocation of memory, which won't be freed until after the hook executes.

In order to avoid this, produce the output only once, save it to memory, and iterate using the copy. For example:

```python
vnl = []
vni = pbs.server().vnodes()
for vn in vni:
    pbs.logmsg(pbs.LOG_DEBUG, "found vn.name=%s" % (vn.name))
    vnl.append(vn)
```
The following functions in PBS Python hooks return iterators, and should be used carefully:

- Iterate over a list of jobs:
  
  ```python
  pbs.server().jobs()
  pbs.queue.jobs()
  ```

- Iterate over a list of queues:
  
  ```python
  pbs.server().queues()
  ```

- Iterate over a list of vnodes:
  
  ```python
  pbs.server().vnodes()
  ```

- Iterate over a list of reservations:
  
  ```python
  pbs.server().resvs()
  ```

### 6.15.9.7 Testing Vnode State

To see whether a vnode has a particular state set:

```python
If v.state == pbs.ND_OFFLINE:
    pbs.logmsg(pbs.LOG_DEBUG, “vnode %s is offline!” % (v.name))
```

### 6.15.9.8 Warnings About Changing Attributes

#### 6.15.9.8.i When to Change Reservation Attributes

The only time that a reservation’s attributes can be altered is during the creation of that reservation.

#### 6.15.9.8.ii Caution About Unsetting Reservation walltime Resource

The `walltime` resource is used to determine the reservation’s `reserve_duration` parameter when the reservation’s `reserve_duration` attribute is not set or is set to `None`. If a `resvsub` hook attempts to unset the `walltime` parameter, for example:

```python
pbs.event().resv.Resource_List["walltime"] = None
```

This will result in the following error:

```bash
% pbs_rsub -R 1800 -l ncpus=1
pbs_rsub: Bad time specification(s)
```
6.15.9.8.iii Changing Job Attributes for a Running Job

When a job is running, only the cput and walltime attributes can be modified. Attempting to change any other attributes for a running job will cause the corresponding qalter action to be rejected. For example, if the job is running, this line in a hook will cause qalter to be rejected:

```python
pbs.event().job.Resource_List["mem"] = pbs.size("10mb")
```

To avoid having the qalter action rejected, check to see whether the job is running, and follow up accordingly. For example:

```python
e = pbs.event()
if int(str(e.job.job_state)) in [ pbs.JOB_STATE_RUNNING, 
    pbs.JOB_STATE_EXITING, pbs.JOB_STATE_TRANSIT ]:
e.accept()
```

6.15.9.8.iv Do Not Unset Array Job Indices

Do not unset pbs.event().job.array_indices_submitted for an array job in a modifyjob hook. For example:

```python
pbs.event().job.array_indices_submitted = None
```

If the hook script is executed for a job array, the qalter request will fail with the message:

```
Cannot modify attribute while job running <job array ID>
```

6.15.9.8.v Do Not Create Job or Reservation Variable List

Hooks are not allowed to create job or reservation Variable_List attributes. Hooks can modify the existing Variable_List job attribute which is supplied by PBS, by modifying values in the list. The following are disallowed in a hook:

```python
pbs.event().job.Variable_List = dict()
pbs.event().resv.Variable_List = dict()
```

These calls will cause the following exception:

```
04/07/2008 11:22:14;0001;Server@host2;Svr;Server@host2;PBS server internal error (15011) in Error evaluating Python script, attribute 'Variable_List' cannot be directly set.
```

To modify the Variable_List attribute:

```python
pbs.event().job.Variable_List["SIMULATE"] = "HOOK00"
```
6.15.9.8.8 Changing Vnode state Attribute

A vnode's state can be set within a runjob hook only if the runjob hook execution concludes with a `pbs.event().reject()` call. This means that if a statement that sets a vnode’s state appears in a runjob hook script, it takes effect only if the following is the last line to be executed:

```plaintext
pbs.event().reject()
```

To set a vnode's state, the syntax is one of the following:

```plaintext
pbs.vnode.state = <vnode state constant>
pbs.vnode.state += <vnode state constant>
pbs.vnode.state -= <vnode state constant>
```

where `<vnode state constant>` is one of the constant objects listed in section 6.13.8.8, “Vnode State Objects”, on page 490.

Examples of changing a vnode’s state attribute:

- To offline a vnode:
  ```plaintext
  pbs.vnode.state = pbs.ND_OFFLINE
  ```

- To add another value to the list of vnode states:
  ```plaintext
  pbs.vnode.state += pbs.ND_DOWN
  ```

- To remove a value from the list of vnode states:
  ```plaintext
  pbs.vnode.state -= pbs.ND_OFFLINE
  ```

When a vnode’s state attribute has no states set, the vnode’s state is equivalent to `free`. This means that you can remove all values, and the vnode will become `free`.

When a vnode’s state is successfully set, the following message is displayed and logged at event class 0x0004:

```plaintext
Node;<vnode-name>;attributes set: state - <vnode state constant> by <hook_name>
```

You can set a vnode’s state attribute in any execution hook and in a periodic hook, and changes to vnode attributes take effect whether the execution hook or periodic hook calls `accept()` or `reject()`.

6.15.9.8.9 Attribute Change Failure is Silent

If you attempt to change the value for an attribute in an unsupported way, PBS does not warn you that your attempt failed.
6.15.9.8.viii Lengthened walltime Can Interfere with Reservations
If a hook lengthens the walltime of a running job, you run the risk that the new walltime will interfere with existing reservations etc.

6.15.10 Local Server Only
Hooks cannot access a server other than the local server. Hooks also cannot specify a non-default server. So for example if a job submission specifies a queue at a server other than the default, the hook can allow that submission, or can change it to the default server, but cannot change it to another non-default server.

6.15.11 Scheduling Impact of Hooks

6.15.11.1 Effect of runjob Hooks on Preemption
With preemption turned on, the scheduler preempts low-priority jobs to run a high-priority job. If the high-priority job is rejected by a runjob hook, then the scheduler undoes the preemption of the low-priority jobs. Suspended jobs are resumed, and checkpointed jobs are restarted.

6.15.11.2 Effect of runjob Hooks with Strict Ordering
When strict_ordering is set to True and backfill is set to False, a most-deserving job that is repeatedly rejected by a runjob hook will prevent other jobs from being able to run. A well-written hook would put the job on hold or requeue the job with a later execution time to prevent idling the system.

6.15.11.3 Effect of runjob Hooks with round_robin and by_queue
With round_robin and by_queue set to True, a job continually rejected by a runjob hook may prevent other jobs from the same queue from being run. A well-written hook would put the job on hold or requeue the job with a later execution time to allow other jobs in the same queue to be run.

A runjob hook's performance directly affects the responsiveness of the PBS scheduler. Consider carefully the trade-off between the work such a hook needs to do and your scheduler's required performance.
6.15.1.4 Peer Scheduling and Hooks

When a job is pulled from one complex to another, the following happens:

- Hooks are applied at the new complex as if the job had been submitted locally
- Any `movejob` hooks at the furnishing server are run

6.15.1.5 Performance Considerations

6.15.1.5.i Cost of Accessing Data

- Using `pbs.server()` to get data about server, queues, jobs, vnodes, or reservations can be slow if run in an execution hook. This is because of the overhead involved when the function has to directly connect to the server and pass requests (via TCP).
- Making queries to `pbs.server().resources_available[]` can be slow.

6.15.1.5.ii Cost of Different Hooks

- Any `queuejob` hooks execute once per job submission
- Any `runjob` hooks execute once per attempt to run a job, after the scheduler has found a place for it
- Any `modifyjob` hooks execute relatively often:
  - Every time a comment is set
  - When a job’s eligibility to run changes
  - Etc.

What this means to the hook writer:

- Your `queuejob` hooks can generally get away with longer run times
- Any hook that needs to listen to `queuejob` events needs to be quick to decide whether it is needed or not

For a fast hook, avoid these:

- Running external commands
- Network connections
- File I/O and logging
- Storing information in server or vnode settings
- Using `pbs.server().resources_available`
- Iterating over the entire set of vnodes() or jobs().
6.15.12 When You Need Persistent Data

If you need your data to be persistent, your hook(s) must be able to save and retrieve the information.

6.15.13 Setting Up Job Environment on Sisters

If you need to set up the job’s environment on sister MoMs, use an `execjob_begin` hook. This hook can set up the desired environment on sister MoMs so that the job can use the new environment.

Here is the order in which hooks and job tasks run:

1. The `execjob_begin` hook runs on all of a job’s sister MoMs, before any of the job’s tasks begin execution.
2. If the job uses `pbs_attach`, job tasks are created on sister MoMs.
3. The `execjob_prologue` hook runs on each sister MoM, but only for the first of the job’s tasks on that MoM.
4. If the job uses `tm_spawn`, job tasks are created on sister MoMs.

The old-style prologue runs only on the Mother Superior; you cannot use it to set up the environment on sister MoMs.

When a job runs on multiple tasks on separate vnodes managed by one MoM, all of the tasks get the same environment.

6.16 Python Modules and PBS

When you run `pbs_python` outside of a hook, `pbs_python` has access to the modules in `pbs_python` and in another PBS-supplied set of modules.

When you run `pbs_python` in a hook, the hook has access only to the modules in `pbs_python`, unless you specify other modules in the hook.

PBS Python modules are in the following directory:

```
PBS_EXEC/lib/python
```
The extended set of modules are in the following directories:

- PBS_EXEC/python/lib/python25.zip
- PBS_EXEC/python/lib/python2.5
- PBS_EXEC/python/lib/python2.5/plat-linux2
- PBS_EXEC/python/lib/python2.5/lib-tk
- PBS_EXEC/python/lib/python2.5/lib-dynload
- PBS_EXEC/python/lib/python2.5/site-packages

If you need to use other modules, we recommend that you put the modules in a different directory from the PBS Python modules.

To use other modules besides the ones in `pbs_python`, specify the path in the hook.

If you are adding modules that are not in PBS Python, you can do this:

```python
import sys
if '/usr/lib64/python2.6' not in sys.path:
    sys.path.append('/usr/lib64/python2.6')
import pbs
```

If you need to include the PBS Python paths, you can do the following. For example, if you put a module in `/usr/lib64/python2.6`, and you want to include the extended set of modules, and `PBS_EXEC` is `/opt/pbs/default`, add them to your hook this way:

```python
import sys
sys.path = ['/usr/lib64/python2.6',
           '/opt/pbs/default/python/lib/python25.zip',
           '/opt/pbs/default/python/lib/python2.5',
           '/opt/pbs/default/python/lib/python2.5/plat-linux2',
           '/opt/pbs/default/python/lib/python2.5/lib-tk',
           '/opt/pbs/default/python/lib/python2.5/lib-dynload',
           '/opt/pbs/default/python/lib/python2.5/site-packages']
for my_path in my_paths:
    if my_path not in sys.path:
        sys.path.append(my_path)
import pbs
```
Chapter 6

Hooks

6.16.1 List of Modules in pbs_python

The following are the modules that are available via pbs_python:

Table 6-22: Modules in pbs_python

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Module Name</th>
<th>Module Name</th>
<th>Module Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>cElementTree</td>
<td>handler</td>
<td>pulldom</td>
<td>stringprep</td>
</tr>
<tr>
<td>collections</td>
<td>handlers</td>
<td>pydoc</td>
<td>symbol</td>
</tr>
<tr>
<td>config</td>
<td>itertools</td>
<td>re</td>
<td>time</td>
</tr>
<tr>
<td>copy</td>
<td>linecache</td>
<td>repr</td>
<td>token</td>
</tr>
<tr>
<td>copy_reg</td>
<td>math</td>
<td>saxutils</td>
<td>tokenize</td>
</tr>
<tr>
<td>cStringIO</td>
<td>md5</td>
<td>select</td>
<td>types</td>
</tr>
<tr>
<td>datetime</td>
<td>minicompat</td>
<td>sre</td>
<td>UserDict</td>
</tr>
<tr>
<td>dis</td>
<td>minidom</td>
<td>sre_compile</td>
<td>UserList</td>
</tr>
<tr>
<td>domreg</td>
<td>NodeFilter</td>
<td>sre_constants</td>
<td>UserString</td>
</tr>
<tr>
<td>ElementInclude</td>
<td>opcode</td>
<td>sre_parse</td>
<td>uuid</td>
</tr>
<tr>
<td>ElementPath</td>
<td>operator</td>
<td>stat</td>
<td>xmlbuilder</td>
</tr>
<tr>
<td>ElementTree</td>
<td>os</td>
<td>statvfs</td>
<td>xmlreader</td>
</tr>
<tr>
<td>expat</td>
<td>parser</td>
<td>string</td>
<td>zipfile</td>
</tr>
<tr>
<td>expatbuilder</td>
<td>pkgutil</td>
<td>StringIO</td>
<td>_exceptions</td>
</tr>
<tr>
<td>expatreader</td>
<td>posixpath</td>
<td>stringold</td>
<td>_socket</td>
</tr>
</tbody>
</table>

6.16.2 Python Version

PBS 12.0 uses Python 2.5. However, in some subsequent release of PBS, we will use a later version of Python. Please note that use of Python 2.5 is deprecated.
6.16.3 Caveats for Python Modules

If you change a Python module in a pre-execution event hook (queuejob, movejob, modify-job, runjob), you must restart the server in order to use the new module, because "import" is cached.

6.17 Errors, Logging, and Troubleshooting

6.17.1 The pbs_python Hook Debugging Tool

You can use the pbs_python interpreter that is shipped with PBS for debugging a hook script from the command line by specifying the --hook option. You can perform interactive or non-interactive debugging. For a complete description of pbs_python, see “pbs_python” on page 72 of the PBS Professional Reference Guide.

Without the --hook option, you can use pbs_python to run the Python interpreter that is distributed with PBS Professional.

Usage for pbs_python:

```
```

6.17.2 Caveats For pbs_python

- When you run pbs_python outside of a hook, it has access to the extended set of modules listed in section 6.16, “Python Modules and PBS”, on page 566. When you run pbs_python inside a hook, the hook does not have access to the extended set of modules. See section 6.16, “Python Modules and PBS”, on page 566.

- If you have added any custom resources, be sure to specify the PBS_HOME/server_priv/resourcedef file with the -r option to pbs_python. Make sure you specify the whole path to the file. For example:

```
pbs_python --hook -r $PBS_HOME/server_priv/resourcedef -i <input_file> <hook.py>
```
6.17.3 Using `pbs_python` to Debug Hooks

In order to run the `pbs_python` tool in its simplest form, use the following basic input file:

```bash
$ cat hook.input
pbs.event().type=<hook_event_name>
pbs.get_local_nodename()=<local_node_name>
```

where `<hook_event_name>` is one of the event names: `queuejob`, `modifyjob`, `resvsub`, `movejob`, `runjob`, `execjob_begin`, `execjob_prologue`, `execjob_epilogue`, `execjob_preterm`, `execjob_end`, `exechost_periodic`, and where `<local_node_name>` is the name of the local host.
6.17.4 Examples of Using \texttt{pbs\_python} to Debug Hooks

Example 6-44: Basic periodic hook, with updates to vnodes:

- Input file:
  ```
  % cat hook.input
  pbs.event().type=exechost_periodic
  pbs.event().vnode_list["host1"].state=free
  pbs.get_local_nodename()=host1
  ```

- Hook file:
  ```
  $ cat test.py
  import pbs

  e = pbs.event()

  pbs.event().vnode_list[pbs.get_local_nodename()].resources_available["ncpus"] = 7
  pbs.event().vnode_list[pbs.get_local_nodename()].resources_available["mem"] = pbs.size("7gb")
  ```

- Run:
  ```
  $ pbs\_python --hook -i hook.input test.py
  pbs.event().accept=True
  pbs.event().reject=False
  pbs.event().vnode_list["host1"].resources_available[ncpus]=7
  pbs.event().vnode_list["host1"].resources_available[mem]=7gb
  ```

Example 6-45: A queuejob hook:

- Input file:
  ```
  $ cat qjob.input
  pbs.event().hook_name=qjob
  pbs.event().hook_type=site
  pbs.event().type=queuejob
  pbs.event().requestor=user1
  pbs.event().requestor_host=host1
  pbs.event().hook_alarm=40
  pbs.event().job.Job_Name=pact
  pbs.event().job.Resource_List[ncpus]=1
  ```
Chapter 6  

Hooks

• Hook file:

```python
$ cat qjob.py
import pbs

e = pbs.event()

e.job.Priority = 7
e.job.Account_Name = "mammoth"
e.job.Resource_List["ncpus"] = 5
e.job.Resource_List["mem"] = pbs.size("5gb")
```

• Run:

```
% pbs_python --hook -i qjob.input qjob.py
pbs.event().accept=True
pbs.event().reject=False
pbs.event().job.Priority=7
pbs.event().job.Resource_List[ncpus]=5
pbs.event().job.Resource_List[mem]=5gb
pbs.event().job.Account_Name=mammoth
```

Example 6-46: Reservation hook:

• Input file:

```
% cat rsub.input
pbs.event().hook_name=qjob
pbs.event().hook_type=site
pbs.event().type=resvsub
pbs.event().requestor=user1
pbs.event().requestor_host=host1
pbs.event().hook_alarm=40
pbs.event().resv.Reserve_Name=my_resv
pbs.event().resv.Resource_List[ncpus]=1
pbs.event().resv.Resource_List[mem]=1mb
```

• Hook file:

```
% cat rsub.py
import pbs
```
def print_attribs(pbs_obj):
    for a in pbs_obj.attributes:
        v = getattr(pbs_obj, a)
        if v and str(v) != "":
            pbs.logmsg(pbs.LOG_DEBUG, "%s = %s" % (a,v))

    e = pbs.event()
    r = e.resv

    print_attribs(r)

    r.Resource_List["select"] = pbs.select("1:ncpus=1:mem=5mb")
    r.Resource_List["place"] = pbs.place("pack:shared")

    # group_list = pbs.group_list
    r.group_list = pbs.group_list("Everyone,Everyone@host2,group1@jobim")

    # Mail_Points= pbs.mail_points
    r.Mail_Points = pbs.mail_points("a")

    # User_List = pbs.user_list
    r.User_List =  pbs.user_list("pbstest,pbstest@host2")

    # Authorized_Users = pbs.acl
    r.Authorized_Users =  pbs.acl("pbstest,user1,Administrator")

    # Authorized_Groups = pbs.acl
    r.Authorized_Groups =  pbs.acl("Everyone,group1,group2")

    • Run:
      % pbs_python --hook -i rsub.input rsub.py
      pbs.event().accept=True
      pbs.event().reject=False
      pbs.event().resv.group_list=Everyone,Everyone@host2,group1@jobim
      pbs.event().resv.User_List=pbstest,pbstest@host2
      pbs.event().resv.Resource_List[select]=1:ncpus=1:mem=5mb
Example 6-47: A modifyjob hook:

- **Hook script:**

  ```
  $ cat modifyjob.py
  
  import pbs

  def print_attribs(pbs_obj):
    for a in pbs_obj.attributes:
      v = getattr(pbs_obj, a)
      if v and str(v) != "":
        pbs.logmsg(pbs.LOG_DEBUG, "%s = %s" % (a,v))
  
  e = pbs.event()

  pbs.logmsg(pbs.LOG_DEBUG, "-------- printing job %s" % (e.job_o.id))
  print_attribs(e.job_o)
  e.job.Priority = 5
  e.job.Resource_List["file"] = pbs.size("7gb")
  e.job.Variable_List["FILE"] = "7gb"
  ```

- **Try with the pbs_python debugging tool:**

  Ensure you have the following input file:

  ```
  % cat hook.input
  
  pbs.event().type=modifyjob
  pbs.event().job.id=0.host1
  pbs.event().job.Variable_List=A=b
  ```

- **Run the hook:**

  ```
  % pbs_python --hook -i hook.input modifyjob.py
  ```

- **The following are printed:**
pbs.event().accept=True  
pbs.event().reject=False  

pbs.event().job.Variable_List=A=b,FILE=7gb  
pbs.event().job.Priority=5  
pbs.event().job.Resource_List[file]=7gb  

• The pbs_python log file shows:

```bash
% cat <yyyyymmdd>
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;----------------->
    printing job 0.host1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;qtime = 1357387083
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Error_Path =
    host1.example.com:/home/user1/STDIN.e0
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;job_state = 1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;schedselect =1:ncpus=1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;ctime = 1357387083
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Rerunnable = 1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;server = host1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;egroup = pbs
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Variable_List
    =A=b,FILE=7gb
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Checkpoint = u
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;etime = 1357387083
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;queue = workq
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Job_Name = STDIN
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;comment = Not Running:
    Could not run job - unable to obtain 1 cpu licenses. avail_licenses=0
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;substate = 10
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;queue_rank = 1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;euser = user1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Mail_Points = a
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Priority = 0
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;project =
```

PBS Professional 12.2 Administrator’s Guide 575
Example 6-48: A movejob hook which prints job ID, src_queue, and movejob event parameters, and sets src_queue:

- Hook script:

```python
$ cat movejob.py

import pbs

def print_attribs(pbs_obj):
    for a in pbs_obj.attributes:
        v = getattr(pbs_obj, a)
        if v and str(v) != ""
            pbs.logmsg(pbs.LOG_DEBUG, "%s = %s" % (a,v))

e = pbs.event()
pbs.logmsg(pbs.LOG_DEBUG, "----------------> printing src_queue %s" % 
(e.src_queue.name))
print_attribs(e.src_queue)
pbs.logmsg(pbs.LOG_DEBUG, "----------------> printing job %s" % 
(e.job.id))
print_attribs(e.job)
```
e.job.queue = pbs.server().queue("workq2")

- Try with the pbs_python debugging tool:

Use the following input file:

```
% cat hook.input2
pbs.event().type=movejob
pbs.event().job.id=<existing-job-id>
```

where `<existing-job-id>` must be some arbitrary job currently existing in the 'workq'. Submit one (qsub -h) if it doesn't exist.

- Run the hook:

```
% pbs_python --hook -i hook.input2 movejob.py
```

- The following is printed:

```python
pbs.event().accept=True
pbs.event().reject=False
pbs.event().src_queue=workq2
```

- The pbs_python log file shows:

```
% cat yyyyymmdd
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;----------------->
printing
src_queue workq
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;name = workq
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;----------------->
printing
job 0.host1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;qtime = 1357387083
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Error_Path =
host1.example.com:/home/user1/STDIN.e0
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;job_state = 1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;qtime = 1357387083
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Rerunable = 1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;server = host1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Variable_List =
```
Chapter 6  

**Hooks**

PBS_O_SYSTEM=Linux, PBS_O_SHELL=/bin/bash, PBS_O_HOME=/home/user1, PBS_O_HOST=host1.example.com, PBS_O_LOGNAME=user1, PBS_O_WORKDIR=/home/user1, PBS_O_LANG=en_US.UTF-8, PBS_O_PATH=/opt/pbs/default/bin:/opt/pbs/default/python/bin:/opt/pbs/default/tcltk/bin:/home/user1/bin:/opt/pbs/default/bin:/opt/pbs/default/python/bin:/opt/pbs/default/tcltk/bin:/home/user1/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/user1/bin:/usr/local/rational/releases/purify.i386_linux2.2003a.06.15.FixPack.0194:/usr/local/purify/base/cots/flexlm.10.8.0.1/i386_linux2:/home/user1/PbsTestLab/bin:/home/user1/bin:/home/user1/bin:/usr/local/rational/releases/
Hooks

Example 6-49: A runjob hook:

- Hook script:

```python

$ cat runjob.py

import pbs
import time

def print_attribs(pbs_obj):
```

PBS Professional 12.2 Administrator’s Guide 579
for a in pbs_obj.attributes:
    v = getattr(pbs_obj, a)
    if v and str(v) != "":
        pbs.logmsg(pbs.LOG_DEBUG, "%s = %s % (a,v))

e = pbs.event()

pbs.logmsg(pbs.LOG_DEBUG, "------------------> printing job %s % \
(e.job.id))
print_attribs(e.job)
e.job.Hold_Types = pbs.hold_types("us")
e.job.Execution_Time = time.mktime([15, 11, 28, 14, 10, 15, -1, -1, 01])
e.job.project="looper"

pbs.event().reject("not allowed to run at this time!")

• Use the following input file:
  % cat hook.input3
  pbs.event().type=runjob
  pbs.event().job.id=<existing-job-id>

where <existing-job-id> must be some arbitrary job currently existing in the server. Submit one (qsub -h) if it doesn't exist.

• Run the hook:
  # pbs_python --hook -i hook.input3 runjob.py

• The following is printed:
  pbs.event().reject=True
  pbs.event().accept=False
  pbs.event().reject_msg=not allowed to run at this time!
  pbs.event().job.Execution_Time=1448745015
  pbs.event().job.Hold_Types=us
  pbs.event().job.project=looper

• The pbs_python log file shows:

  % cat <yyyymmdd>
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;----------------->
  printing
  job 5.host1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;qtime = 1357424154
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Error_Path =
  host1.example.com:/home/user1/bugs/sp260361/STDIN.e5
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;job_state = 2
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;ctime = 1357424154
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Rerunable = 1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;server = host1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Variable_List =
  PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,PBS_O_HOME=/home/
    user1,PBS_O_HOST=host1.example.com,PBS_O_LOGNAME=user1,PBS_O_WORKDIR=/
    home/user1/bugs/sp260361,PBS_O_LANG=en_US.UTF-8,PBS_O_PATH=/opt/pbs/
    default/bin:/opt/pbs/default/python/bin:/opt/pbs/default/tcltk/bin:/
    home/user1/bin:/opt/pbs/default/bin:/opt/pbs/default/python/bin:/opt/
    pbs/default/tcltk/bin:/home/user1/bin:/usr/local/bin:/bin:/usr/bin:/
    usr/X11R6/bin:/home/user1/bin:/usr/local/rational/releases/
    purify.i386_linux2.2003a.06.15.FixPack.0194:/usr/local/purify/base/
    cots/flexlm.10.8.0.1/i386_linux2:/home/user1/PbsTestLab/bin:/home/
    user1/bin:/home/user1/bin:/usr/local/rational/releases/
Hooks

purify.i386_linux2.2003a.06.15.FixPack.0194:/usr/local/purify/base/cots/flexlm.10.8.0.1/i386_linux2:/home/user1/PbsTestLab/bin,PBS_O_QUEUE=workq,PBS_O_MAIL=/var/spool/mail/user1

01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Checkpoint = u
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Submit_arguments = <jsdl-hpcpa:Argument>-h</jsdl-hpcpa:Argument>
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;queue = workq
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Job_Name = STDIN
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;substate = 20
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Mail_Points = a
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Priority = 0
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;project = _pbs_project_default
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Output_Path = host1.example.com:/home/user1/bugs/sp260361/STDIN.o5
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Hold_Types = u
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Join_Path = n
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;mtime = 1357424154
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;id = 5.host1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Resource_List = select=1:ncpus=1,nodect=1,ncpus=1,place=pack
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Keep_Files = n
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;_connect_server = host1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Job_Owner = user1@host1.example.com
6.17.5 Debugging Hook Scripts

6.17.5.1 Procedure for Hook Debugging

The following steps may help you avoid errors in hook scripts:

1. Create a hook, and import its content.

2. Temporarily set the Server’s `log_events` to a higher value such as 2047 to see plenty of logging.

3. Do a test run of the hook script, by causing events (e.g. `qsub`, `qalter`, `qmove`, `pbs_rsub`) that invoke the hook script. Check for error messages in the server logs.

4. Correct the hook script, re-import the fixed code, and rerun the test.

5. Once the hook script is running fine, then set the Server’s `log_events` back to default (i.e. 511).

6.17.5.2 When Commands Fail Inside Hooks

When a command fails inside a hook, but succeeds outside the hook, the problem may be a difference in the environments.

6.17.6 Error Reporting and Logging

Hook errors are printed to `stderr` for the command (qsub, qalter, pbs_rsub, or qmove) that triggered the hook. If the hook provides a custom error message, that message is treated the same way.

Hooks can log custom strings to the log file of the daemon from which the hook is executing. When logging a message, a hook uses message logging methods to specify the message, and constant objects to specify the log event class. See “`pbs.logmsg(log event class, message)`” on page 534, and See “Message Log Event Class Objects” on page 487.

When the PBS server starts, it prints to the server logs both the Python version integrated with the server, and a list of all the hook names registered with the server.

If you want to see only hook-related MoM log messages, for example “<hook name>; started”, “<hook name>; finished”, set the `$logevent` parameter in the MoM configuration file to 0x400.

To see all the different types of MoM log messages, set `$logevent` to 0xffff.

To see only hook-related 0x0400 messages in the MoM logs, such as “<hook name>; started”, “<hook_name>; finished”, set `$logevent` to 0x400 in the MoM configuration file.
The default value for the $logevent MoM parameter is 975, so that the following log events are captured. See section 12.4.4, “Log Event Classes”, on page 868 for details about log levels.

PBSEVENT_ERROR
PBSEVENT_SYSTEM
PBSEVENT_ADMIN
PBSEVENT_JOB
PBSEVENT_JOB_USAGE
PBSEVENT_SECURITY
PBSEVENT_DEBUG
PBSEVENT_DEBUG2
PBSEVENT_RESV

6.17.7 Errors During Creation and Deployment

6.17.7.1 Hook Name Matches Existing Hook

Creating a hook whose name matches that of an existing hook: the following error message is printed in stderr and in the server logs:

“hook error: hook name <hook_name> already registered, try another name”

6.17.7.2 Using a Hook Name that Starts with “PBS”

Using a hook name that starts with “PBS”: the hook name is rejected with the following error in qmgr’s stderr, as well as in the server logs:

“hook error: cannot use PBS as a prefix - it is reserved for PBS hooks”

6.17.7.3 Deleting a Non-Existent Hook

Deleting a non-existent hook: the following is returned in qmgr’s stderr and server logs:

“qmgr: hook error: <non-existent hook name> does not exist”

6.17.7.4 Specifying a Non-Existent Event Type

Specifying a non-existent event type: an error message is printed to qmgr’s stderr and also to the server logs:
Example:

```
Qmgr: set hook hook1 event="mom_checkpoint"
  "hook error: invalid argument to event. Should be one of: queuejob,
   modifyjob, resvsub, movejob, runjob, provision, execjob_begin,
   execjob_prologue, execjob_epilogue, execjob_preterm, execjob_end,
   exechost_periodic, or "" for no event."
  "qmgr: hook error returned from server"
```

6.17.7.5 Using a Bad Hook Value

Putting in a bad hook value: an error is printed to `qmgr`'s `stderr` and also to the server logs:

Example:

```
Qmgr: set hook hook2 order=1025
  "qmgr obj=hookA svr=default: order given (1025) is outside the acceptable
   range of [1, 1000] for type 'site'."
  "qmgr: hook error returned from server"
```

6.17.7.6 Unauthorized User

If `qmgr` is invoked, and the object being operated on is “hook”, and the executing user at some host does not have access to the target server's private location for hooks data, then the following error is issued to `stderr` and server logs:

```
"<user>@<host> is unauthorized to access hooks data from server <hostname>"
```

6.17.7.7 Setting a Bad Hook Type

Setting a bad type to a hook produces the following error message in `qmgr`'s `stderr` and also in the server logs:

```
"hook error: invalid argument to type. Must be site"
```

6.17.7.8 Setting a Bad Alarm Value

Setting a bad alarm value to a hook produces the following error message in `qmgr`'s `stderr` and also in the server logs:

```
"hook error: alarm value of a hook must be > 0"
```
6.17.7.9 Exporting To Non-Writable File

Exporting a hook's content to a file that is not writable due to ownership or permission problems results in the following error message being printed to stderr:

"qmgr: hook error: <output_file> permission denied"

6.17.7.10 Setting Bad Hook user Attribute

Setting a value for the user attribute of a hook to something other than “pbsadmin” produces the following error message in qmgr's stderr and also in the server logs:

"hook error: user value of a hook must be pbsadmin, pbsuser"

This attribute does not need to be set to the actual name of the PBS service account.

6.17.7.11 Importing From Non-Readable File

Importing a hook where the PBS server is unable to open the input file because the file is non-existent, has a permission problem, or any other system-related error causes the following error message to be printed in stderr and in the server logs:

"qmgr: hook error: unable to open <filename> by server run by <user>@<host>: <error message>"

Examples:

"qmgr: hook error: unable to open hook1.py by server run by pbsadmin@hostX: permission denied"

"qmgr: hook error: unable to open hook1.py by server run by pbsadmin@hostY: No such file or directory"

6.17.7.12 Importing/Exporting with Wrong Content-Type

Importing/exporting a hook where the <content-type> is something other than “application/x-python” causes the following error message to be printed in stderr and in the server logs:

"qmgr: hook error: <content_type> must be 'application/x-python'"

Importing/exporting a hook where the <content-encoding> is something other than “default” or “base64” causes the following error message to be printed in stderr and on the server logs:

"qmgr: hook error: <content_encoding> must be 'default' or 'base64'"
An import call on a hook that already has a content script results in the following informational message being printed in stdout and server logs:
  "qmgr: hook <hook_name> contents overwritten by file <hook input file>

### 6.17.7.13 Setting Vnode State to Invalid Value

Setting a vnode’s state attribute to an invalid value causes the `pbs.BadAttributeValueError` exception to be raised.

### 6.17.7.14 Creating a Hook with Same Name as Existing Hook

You may find that when you remove a hook, it takes some time for the hook to be completely purged. If you run "`qmgr -c 'create hook <hook_name>'`" where a previous hook of the same `<hook_name>` still exists, you will see the following message:

  "hook name <hook_name> is pending delete, try another name"

Either specify another name for the hook, or retry the `qmgr` request again later, after the previous hook is completely purged.

### 6.17.8 Errors And Messages During Hook Execution

#### 6.17.8.1 Successful Operation of runjob Hook

When a hook successfully sets an attribute, one of the following is written to the server’s log:

  `<job ID>; ‘<hook name>’ hook set job’s <attribute name> = <value>`

or

  `Job held by ‘<hook name>’ hook on <timedate>`

#### 6.17.8.2 Unsuccessful Operation for runjob Hook

When a hook fails to set an attribute, the following is written to the server’s log:

  `<job ID>; ‘<hook name>’ hook failed to set job’s <attribute name> = <value>`
6.17.8.3 Rejecting an Action

If a hook rejects an action by calling the `pbs.event().reject()` function:

- The following messages are printed to stderr of the command that triggered the hook:
  
  "<command_name>: Request rejected by filter hook <hook_name>"
  
  "<command_name>: '<msg' value passed to pbs.event().reject()>

  where 'msg' is the message passed (if any) as input to `pbs.event().reject()`.

- The following messages are printed in the appropriate PBS daemon log, logged at event class 0x0400:
  
  "<user>@<host>...<request type> request rejected by <hook name>

  "<user>@<host>...<request type> '<msg' value passed to pbs.event().reject()>

6.17.8.4 Triggering an Alarm

If the alarm was triggered while executing a hook:

- The command that initiated the request gets the following messages in its stderr:
  
  "<command_name>: Request rejected by filter hook <hook_name>"
  
  "<command_name>: alarm call while running hook <hook_name>"

- The following entry appears in the appropriate PBS daemon log, logged under event class 0x0100:
  
  "<user>@<host>...<request type> alarm call while running hook <hook_name>,

  request rejected"

6.17.8.5 Encountering an Unhandled Exception

If a hook encounters an unhandled exception:

- PBS rejects the corresponding action. The command that triggered the hook gets the following message in stderr:
  
  "<command_name>: request rejected as filter hook <hook_name> encountered an exception. Inform admin."

- The following message appears on the appropriate PBS daemon log, logged under 0x0100 event class:
  
  "<request type> hook <hook_name> encountered an exception, request rejected"

See section 6.13.7.2.i, “Unhandled Exceptions”, on page 485.
6.17.8.6 Starting and Finishing Hook Execution

Whenever hook execution starts or finishes, timestamped 0x0400 event class log messages appear in the appropriate PBS daemon log:

"11/13/2007 00:00:42 ...<user>@<host>...<request type> running hook named <hook name>"

"11/13/2007 00:01:42<user@><host>...<request type> <hook_name> finished"


6.17.8.7 Hook Timeout

When a hook timeout is triggered, the hook script gets a Python KeyboardInterrupt from the PBS server. The server logs show the following:

06/17/2008 17:57:16;0001;Server@host2;Svr;Server@host2;PBS server internal error (15011) in Python script received a KeyboardInterrupt, <type 'exceptions.KeyboardInterrupt'>

6.17.8.8 Hooks Attempting I/O

When the PBS server is running, stdout, stderr, and stdin are closed, so that a hook script containing calls to print to standard output or standard error, or to read input from standard input, gets the following exception:

02/24/2008 08:03:34;0086;Server@a-centauri;Svr;Server@a-centauri;Compiling script file: </usr/spool/PBS/server_priv/hooks/hook_test.PY>

02/24/2008 08:03:34;0001;Server@a-centauri;Svr;Server@a-centauri;PBS server internal error (15011) in Error evaluating Python script, <type 'exceptions.IOError'>

6.17.8.9 runjob Hook Errors

6.17.8.9.i Modifying Hold, Execution Time, Dependency, or Project of Accepted Job

runjob hooks can change the Hold_Types, Execution_Time, depend, and project attributes only for jobs that are rejected by the hook. If a runjob hook accepts an event request, using pbs.event().accept(), but attempts to do any of the above, then the hook request is rejected.
If the hook is triggered by a `qrun` command, the following message is sent to `stderr` where the `qrun` command was run. If the hook is triggered when the scheduler tries to run the job, the following message is written to the job’s `comment` attribute:

```
request rejected by filter hook <hook_name>: cannot modify job after runjob request has been accepted.
```

The following message is written to the PBS server log, at log event class `0x0100`:

```
<hook name>; Found job <attribute name> attribute flagged to be set runjob request rejected by <hook name>: cannot modify job after runjob request has been accepted.
```

### 6.17.8.9.ii Modifying Output or Error Path of Rejected Job

Runjob hooks can change the `Output_Path` and `Error_Path` attributes only for jobs that are accepted by the hook. If a `runjob` hook rejects an event request, using `pbs.event().reject()`, but attempts to do any of the above, the following message is written to the PBS server log, at log event class `0x0100`:

```
runjob request rejected by <hook name>: cannot modify job attribute <attribute name> after runjob request has been rejected.
```

### 6.17.8.9.iii Modifying Vnode

If a `runjob` hook event is accepted via a `pbs.event().accept()` call, and yet an attempt is made to modify a vnode's state, then the hook request is rejected. The following message is sent to the `stderr` of `qrun`, and becomes the job’s comment:

```
request rejected by filter hook <hook_name>: cannot modify vnode after runjob request has been accepted.
```

The following message appears in the PBS server log, logged at event class `0x0100`:

```
runjob request rejected by <hook name>: cannot modify a vnode after runjob request has been accepted.
```

### 6.17.8.9.iv Runjob Hook Referencing Wrong Parameter

If a `runjob` hook attempts to reference a `pbs.event()` parameter other than `pbs.event().job`, the exception `pbs.EventIncompatibleError` is raised.

### 6.17.8.9.v Attempting to Set Restricted Resource

A `runjob` hook cannot set the value of a `Resource_List` member other than those listed in section 6.13.5.4.i, “Restrictions on Using Job’s Resource List Attribute by Hooks”, on page 469.
Setting any of the wrong resources results in the following:

- The hook request is rejected
- The following message is sent to the STDERR of `qrun`, or as output from calling `pbs_geterrmsg()` after the failed `pbs_runjob()`:

  "request rejected by filter hook: '<hook name>' hook failed to set job's 
  Resource_List.<resc_name> = <resc_value> (not allowed)"

- The scheduler updates the affected job's comment attribute with the above message.
- The following message appears in the server's log, logged at level 0x0100:

  "runjob request rejected: '<hook name>' hook failed to set job's 
  Resource_list.<resc_name> = <resc_value> (not allowed)"

### 6.17.8.10 Special Errors Requiring Support

If you encounter any of the following log messages, an internal failure has occurred during hook setup. Please contact PBS Professional support:

- 04/15/2011 17:55:23;0100;Server@jobim;Hook;<hook_name>t3;Encountered an 
  error while setting event
- 04/15/2011 17:55:23;0001;Server@jobim;Svr;Server@jobim;PBS server internal 
  error (15011) in _get_job, partially populated python job object
- 04/15/2011 17:55:23;0001;Server@jobim;Svr;Server@jobim;PBS server internal 
  error (15011) in _get_server, partially populated python server object
- 04/15/2011 17:55:26;0001;Server@jobim;Svr;Server@jobim;PBS server internal 
  error (15011) in _get_queue, partially populated python queue object
- 04/15/2011 17:55:26;0001;Server@jobim;Svr;Server@jobim;PBS server internal 
  error (15011) in _get_vnode, warning: partially populated python vnode object
- 04/15/2011 17:55:26;0001;Server@jobim;Svr;Server@jobim;PBS server internal 
  error (15011) in _get_resv, warning: partially populated python resv object

### 6.17.9 Errors During Startup

If the server starts up and encounters a hook that has no content (no script was imported into the hook), PBS displays the following warning:

"failed to stat <path_server_priv_hooks>/<hook_name>.PY"
"failed to allocate storage for python script
<path_server_priv_hooks>/<hook_name>.PY"
6.17.10 Errors in Hook Updates

- If a job runs where a hook update is incomplete, PBS prints the following to the server’s log file:
  "vnode <node_name>'s parent mom <mom_host>:<mom_port> has a pending copy hook or delete hook request"

Bear in mind that hooks are updated asynchronously with respect to jobs, so a multi-host job that started before the update may encounter an incompletely updated hook. See section 6.5, “Updating Hooks and Custom Resources”, on page 419.

- As PBS copies or deletes execution or periodic hooks to the MoMs, the following messages are printed in the server’s log file at 2047:
  "successfully sent hook file <filename> to <mom_hostname>"
  "successfully sent rescdef file <filename> to <mom host name>"
  "successfully deleted hook file <filename> from <mom host name>"
  "successfully deleted rescdef file <filename> from <mom host name>"
  "failed to copy hook file <filename> to <mom host name>"
  "failed to copy rescdef file <filename> to <mom host name>"
  "failed to delete hook file <filename> from <mom host name>"
  "failed to delete rescdef file <filename> from <mom host name>"

- You may find that when you remove a hook, it takes some time for the hook to be completely purged. If you run "qmgr -c 'create hook <hook_name>'" where a previous hook of the same <hook_name> still exists, you will see the following message:
  "hook name <hook_name> is pending delete, try another name"

Either specify another name for the hook, or retry the qmgr request again later, after the previous hook is completely purged.

6.17.11 Hook-related Error Codes

The following are hook-related error codes:

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBSE_MOM_INCOMPLETE_HOOK</td>
<td>15167</td>
<td>Execution hooks not fully transferred to a particular MoM</td>
</tr>
</tbody>
</table>
### 6.17.12 Troubleshooting

#### 6.17.12.1 Bad Interpreter Path

If you see the following error:

```
/opt/pbs/default/bin/pbs_python: bad interpreter: No such file or directory
```

You should check to see whether this is a valid path on this host. Try to `cd` to the job execution directory and execute any command using this interpreter path.

#### 6.17.12.2 Viewing Hook Propagation

You don’t need to restart `pbs_mom` for a MoM hook to take effect. If you use `qmgr`, the server takes care of copying the new hook over to the MoM, in the background. It’s possible a job may have seen the old MoM hook before the new hook arrives. But eventually, the hook will get there. You’ll see a message in the `server_logs` with the following:

```
vnode <name>‘s parent mom <mom_name> has a pending copy hook or delete hook request
```

### Table 6-1: Hook-related Error Codes

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBSE_MOM_REJECT_ROOT_SCRIPTS</td>
<td>15168</td>
<td>A MoM has rejected a request to copy a hook-related file, or a job script to be executed by root</td>
</tr>
<tr>
<td>PBSE_HOOK_REJECT</td>
<td>15169</td>
<td>A MoM received a reject result from a mom hook</td>
</tr>
<tr>
<td>PBSE_HOOK_REJECT_RERUNJOB</td>
<td>15170</td>
<td>Hook rejection requiring a job to be rerun</td>
</tr>
<tr>
<td>PBSE_HOOK_REJECT_DELETEJOB</td>
<td>15171</td>
<td>Hook rejection requiring a job to be deleted</td>
</tr>
</tbody>
</table>
6.18 Attributes and Parameters Affecting Hooks

- Each hook’s attributes affect the behavior of that hook. Hook attributes are listed in section 6.11.1, “Hook Attributes”, on page 446.
- The $reject_root_scripts MoM parameter controls whether MoM accepts new hook scripts.
- The topology_info vnode attribute shows topology information, and is visible only in hooks.

6.19 See Also

For a description of the PBS hook APIs, see the PBS Professional Programmer’s Guide. Each PBS object’s attribute’s Python type is listed in its description in “Attributes” on page 311 of the PBS Professional Reference Guide. For example, “Server Attributes” on page 316 of the PBS Professional Reference Guide lists the Python type for the job_sort_formula server attribute.

The following man pages and equivalent sections contain useful information:

Table 6-23: See Also

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Guide Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs_module(7B)</td>
<td>section 9.3.1, “The pbs Module”, on page 107 of the PBS Professional Programmers Guide</td>
</tr>
<tr>
<td>pbs_stathook(3B)</td>
<td>section 9.3.3, “The pbs_stathook() API”, on page 125 of the PBS Professional Programmers Guide</td>
</tr>
<tr>
<td>pbs_hook_attributes(7B)</td>
<td>“Hook Attributes” on page 396 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td>pbs_job_attributes(7B)</td>
<td>“Job Attributes” on page 375 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td>pbs_server_attributes(7B)</td>
<td>“Server Attributes” on page 316 of the PBS Professional Reference Guide</td>
</tr>
</tbody>
</table>
### Table 6-23: See Also

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Guide Section</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pbs_queue_attributes(7B)</code></td>
<td>“Queue Attributes” on page 354 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>pbs_node_attributes(7B)</code></td>
<td>“Vnode Attributes” on page 366 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>qmgr(1B)</code></td>
<td>“qmgr” on page 150 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>qsub(1B)</code></td>
<td>“qsub” on page 211 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>qmove(1B)</code></td>
<td>“qmove” on page 172 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>qalter(1B)</code></td>
<td>“qalter” on page 127 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>pbs_rsub(1B)</code></td>
<td>“pbs_rsub” on page 78 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>pbs_manager(3B)</code></td>
<td>section, “pbs_manager”, on page 42 of the PBS Professional Programmers Guide</td>
</tr>
</tbody>
</table>
Chapter 6  

**Hooks**
Chapter 7

Provisioning

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application.

7.1 Introduction

You can configure vnodes so that PBS will automatically install the OS or application that jobs need in order to run on those vnodes. For example, you can configure a vnode that is usually running RHEL to run SLES instead whenever the Physics group runs a job requiring SLES. If a job requires an application that is not usually installed, PBS can install the application in order for the job to run.

You can use provisioning for booting multi-boot systems into the desired OS, downloading an OS to and rebooting a diskless system, downloading an OS to and rebooting from disk, instantiating a virtual machine, etc. You can also use provisioning to run a configuration script or install an application.

7.2 Definitions

AOE
The environment on a vnode. This may be one that results from provisioning that vnode, or one that is already in place

Master Provisioning Script, Master Script
The script that makes up the provisioning hook
Chapter 7  Provisioning

Provision
To install an OS or application, or to run a script which performs installation and/or setup

Provisioning Hook
The hook which performs the provisioning, either by calling other scripts or running commands

Provisioning Tool
A tool that performs the actual provisioning, e.g. SGI Tempo

Provisioned Vnode
A vnode which, through the process of provisioning, has an OS or application that was installed, or which has had a script run on it

7.3  How Provisioning Can Be Used

• Each application requires specific version of OS
  The site runs multiple applications, and each application may be certified to run on a specific OS. In this situation, a job that will run an application requiring a specific OS requests the OS, and PBS provisions the required OS.
• The site needs differently configured images of the same OS to be loaded at different times
  The site has multiple projects, and each project requires the OS to be configured in a different way on a group of hosts. In this situation, PBS provisions groups of hosts with the correct OS image, for the time period needed by each project. The OS image is configured and supplied by the site administrator.
• The entire site needs different OSes at different times of day
  The entire site runs one OS during certain hours, and a different OS at other times. For example, the site runs Windows during the day to support users running interactive Excel sessions, and runs Linux at night for batch jobs. The site does not run any PBS MoMs during the day.
• A user reserves multiple vnodes running the same version of an OS
  A user may need a specific version of an OS for a period of time. For example, a user needs 5 nodes running RHEL3 from 5pm Friday until 5am Monday.
• The administrator wants to limit the number of hosts that are being provisioned at any
one time, for any of the following reasons:
- The network can become overwhelmed transferring OS images to execution nodes
- The hosts can draw excessive power if many are powering up at the same time
- Some sites notify the administrator whenever an execution node goes down, and when several vnodes are provisioned, the administrator is paged repeatedly

7.4 How Provisioning Works

7.4.1 Overview of Provisioning

PBS allows you to create a provisioning hook, which is a hook that is triggered by a provisioning event. When this hook is triggered, it manages the required provisioning on the vnodes to be provisioned. The hook calls a provisioning mechanism such as SGI’s Tempo to accomplish the provisioning.

Provisioning can be the following:
- Directly installing an OS or application
- Running a script which may perform setup or installation

PBS allows you to configure each vnode with a list of available AOE.s. This list is specified in the vnode’s `resources_available.aoe` resource. Each vnode’s `current.aoe` attribute shows that vnode’s current AOE. The scheduler queries each vnode’s `aoe` resource and `current.aoe` attribute in order to determine which vnodes to provision for each job.

When users submit jobs, they can request a specific AOE for each job. When the scheduler runs each job, it either finds the vnodes that satisfy the job’s requirements, or provisions the required vnodes.

Users can create reservations that request AOE.s. Each reservation can have at most one AOE specified for it. Any jobs that run in that reservation must not request a different AOE.

7.4.1.1 Rebooting When Provisioning

When provisioning a vnode with some AOE.s, the vnode must be rebooted as part of the provisioning process. Some OS installations, for example, require rebooting. In this case, the provisioning script must cause the vnode to be rebooted.

When the installation does not require a reboot, the provisioning script does not need to cause the vnode to be rebooted. For example, provisioning with some applications does not require a reboot.
7.4.2 How Vnodes Are Selected for Provisioning

Each job can request at most one AOE. When scheduling the job, PBS looks for vnodes with the requested AOE, as with any other resource. If there are not enough vnodes with the requested AOE, PBS tries to provision vnodes in order to satisfy the job’s requirements.

7.4.2.1 Provisioning Policy

PBS allows a choice of provisioning policies. You set the scheduler’s provision_policy configuration parameter to be either “avoid_provision” or “aggressive_provision”. The default provisioning policy is “aggressive_provision”.

avoid_provision

PBS first tries to satisfy the job’s request from free vnodes that already have the requested AOE instantiated. PBS uses node_sort_key to sort these vnodes.

If it cannot satisfy the job’s request using vnodes that already have the requested AOE instantiated, it does the following:

• PBS uses node_sort_key to select the free vnodes that must be provisioned in order to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them.

• Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

aggressive_provision

PBS selects vnodes to be provisioned without considering which AOE is currently instantiated.

PBS uses node_sort_key to select the vnodes on which to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them. Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

7.4.2.2 Examples of Vnode Selection

The following examples show how provisioning policy can affect which vnodes are selected for provisioning.

Example 7-1: 3 vnodes

In sched_config:

    node_sort_key: “ncpus HIGH"
We have 3 nodes as described in the following table:

**Table 7-1: Example Configuration**

<table>
<thead>
<tr>
<th>Vnode/Host</th>
<th>Number of CPUs</th>
<th>Current AOE</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>host1</td>
<td>1</td>
<td>aoe1</td>
<td>free</td>
</tr>
<tr>
<td>host2</td>
<td>2</td>
<td>unset</td>
<td>free</td>
</tr>
<tr>
<td>host3</td>
<td>3</td>
<td>aoe2</td>
<td>free</td>
</tr>
</tbody>
</table>

No jobs are running on any of the vnodes.

Case 1: aggressive provisioning

```
provision_policy: "aggressive_provision"
```

Job submitted with `-lselect=ncpus=1:aoe=aoe1`

In this case, host3 is used to run the job and host3 is provisioned.

Case 2: avoiding provisioning

```
provision_policy: "avoid_provision"
```

Job submitted with `-lselect=ncpus=1:aoe=aoe1`

In this case, host1 is used to run the job and host1 is not provisioned.

Example 7-2: 5 vnodes

The following table shows the example configuration:

**Table 7-2: Example Configuration**

<table>
<thead>
<tr>
<th>Vnode/Host</th>
<th>AOE Available</th>
<th>Current AOE</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>aoe1, aoe2</td>
<td>aoe1</td>
<td>busy</td>
</tr>
<tr>
<td>N2</td>
<td>aoe1, aoe2</td>
<td>aoe2</td>
<td>free</td>
</tr>
<tr>
<td>N3</td>
<td>aoe1, aoe2</td>
<td>NULL</td>
<td>free</td>
</tr>
<tr>
<td>N4</td>
<td>aoe1, aoe2</td>
<td>aoe1</td>
<td>free</td>
</tr>
<tr>
<td>N5</td>
<td>aoe1, aoe2</td>
<td>aoe1</td>
<td>free</td>
</tr>
</tbody>
</table>
The vnodes are sorted in the order N1, N2, N3, N4, N5.

A job is submitted with:

```
qsub -lselect=3:ncpus=1:aoe=aoe1 -lplace=scatter
```

The job needs three vnodes with aoe1. Assume that all other requests except that for the AOE can be satisfied by any vnode.

Case 1: aggressive provisioning

The scheduler selects N2, N3 and N4. It has not considered the AOE instantiated on these vnodes. It then provisions N2 and N3 since N2 has a different AOE instantiated on it and N3 is not provisioned yet. N4 is not provisioned, because it has the requested AOE already instantiated.

Case 2: avoiding provisioning

First, the scheduler selects N4 and N5. It does not choose N2 since it has a different AOE instantiated, and it does not choose N3 since it does not have any AOE instantiated. But N4 and N5 together do not satisfy the job's requirement of 3 vnodes.

Second, the scheduler seeks vnodes that if provisioned can satisfy the job’s request. N2 and N3 can each satisfy the job’s request, so it chooses N2, because it comes first in sorted order.

The job runs on N4, N5 and N2. N2 is provisioned.

7.4.2.3  Rules for Vnode Selection for Provisioning

A vnode is not selected for provisioning for the following reasons:

- It does not have the requested AOE available in its list
- It does not have provisioning enabled on it
- It has other running or suspended jobs
- It already has the requested AOE

7.4.2.4  Triggering Provisioning

When a job requires a vnode, and the vnode’s current_aoe attribute is unset, or is set to a different AOE from the one requested, the vnode is provisioned.
7.4.3 Provisioning And Reservations

7.4.3.1 Creating Reservations that Request AOE

A reservation can request at most one AOE.

When a user creates a reservation that requests an AOE, the scheduler searches for vnodes that can satisfy the reservation. When searching, the scheduler follows the rule specified in the `provision_policy` scheduling parameter in `PBS_HOME/sched_priv/sched_config`. See the `pbs_sched(8B)` manual page.

The vnodes allocated to a reservation that requests an AOE are put in the `resv-exclusive` state when the reservation runs. These vnodes are not shared with other reservations or with jobs outside the reservation.

7.4.3.2 Submitting Jobs to a Reservation

If a job that requests an AOE is submitted to a reservation, the reservation must request the same AOE.

7.4.3.3 Running a Job in a Reservation Having a Requested AOE

A job can run in a reservation that has requested an AOE, as long as the job fits the following criteria:

- It requests the same AOE as the reservation

If the job has requested no AOE, or an AOE different from that of the reservation, the job is rejected.

7.4.4 How Provisioning Affects Jobs

7.4.4.1 Preemption and Provisioning

A job that has requested an AOE will not preempt another job, regardless of whether the job’s requested AOE matches an instantiated AOE. Running jobs are not preempted by jobs requesting AOE.
Chapter 7  Provisioning

7.4.4.2  Backfilling and Provisioning

If the job being backfilled around or the job doing the backfilling share a vnode, a job that has requested an AOE will not play any part in backfilling:

• It will not be backfilled around by smaller jobs.
• It will not be used as the job that backfills around another job.

7.4.4.3  Walltime and Provisioning

A job’s walltime clock is started after provisioning is over.

7.4.4.4  Using qrun

When a job requesting an AOE is run via qrun  -H, the following happens:

• If the requested AOE is available on the specified vnodes, those vnodes are provisioned with the requested AOE
• If the requested AOE is not available on the specified vnodes, the job is held

7.4.5  Vnode States and Provisioning

7.4.5.1  States Associated With Provisioning

The following vnode states are associated with provisioning:

provisioning

A vnode is in the provisioning state while it is in the process of being provisioned. No jobs are run on vnodes in the provisioning state.

wait-provision

There is a limit on the maximum number of vnodes that can be in the provisioning state. This limit is specified in the Server’s max_concurrent_provision attribute. If a vnode is to be provisioned, but cannot because the number of concurrently provisioning vnodes has reached the specified maximum, the vnode goes into the wait-provisioning state. No jobs are run on vnodes in the wait-provisioning state.

resv-exclusive

The vnodes allocated to a reservation that requests an AOE are put in the resv-exclusive state when the reservation runs. These vnodes are not shared with other reservations or with jobs outside the reservation.
7.4.5.2 Provisioning Process

The following table describes how provisioning and vnode state transitions interact:

**Table 7-3: Vnode State Transitions and Provisioning**

<table>
<thead>
<tr>
<th>Event</th>
<th>Starting Vnode State</th>
<th>Ending Vnode State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vnode is selected for provisioning</td>
<td>free</td>
<td>provisioning</td>
</tr>
<tr>
<td>Provisioning on vnode finishes</td>
<td>provisioning</td>
<td>free</td>
</tr>
<tr>
<td>1. Job running on this vnode leaving some resources available</td>
<td>free</td>
<td>free</td>
</tr>
<tr>
<td>2. No job running on this vnode</td>
<td>free</td>
<td>free</td>
</tr>
<tr>
<td>Job running on this vnode, using all resources</td>
<td>free</td>
<td>job-busy</td>
</tr>
<tr>
<td>Vnode is selected for provisioning, but other vnodes being provisioned</td>
<td>free</td>
<td>wait-provisioning</td>
</tr>
<tr>
<td>maximum allowed number of concurrently provisioning vnodes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This vnode is waiting to be provisioned for a multi-vnode job</td>
<td>wait-provisioning</td>
<td>free</td>
</tr>
<tr>
<td>and provisioning fails for another of the job’s vnodes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provisioning fails for this vnode</td>
<td>provisioning</td>
<td>offline</td>
</tr>
<tr>
<td>This vnode is waiting to be provisioned, and another vnode finishes</td>
<td>wait-provisioning</td>
<td>provisioning</td>
</tr>
<tr>
<td>provisioning, bringing the total number of provisioning vnodes below</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the limit specified in max_concurrent_provision</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7.4.5.3 Vnode State When Provisioning Fails

If provisioning fails on a vnode, that vnode is put into the **offline** state.

If provisioning for a multi-vnode job fails on one vnode, any vnodes in the **wait-provisioning** state are put into the **free** state.
7.4.5.4 Using the `qmgr` Command on Vnodes In Process of Provisioning

The following changes cannot be made to a provisioning vnode (a vnode in the provisioning state):

- Changing value of `current_aoe` vnode attribute
- Modifying resource `resources_available.aoe`
- Changing the state of the vnode. The `qmgr` command returns an error if this is attempted.
- Deleting the vnode from the Server. The `qmgr` command returns an error if this is attempted.

The following can be modified while a vnode is provisioning:

- The Server’s `max_concurrent_provision` attribute
- A provisioning vnode’s `provision_enable` attribute

The following cannot be set on the Server’s host:

- `current_aoe` vnode attribute
- `provision_enable` vnode attribute
- `resources_available.aoe`

7.4.6 Attributes, Resources, and Parameters Affecting Provisioning

7.4.6.1 Host-level Resources

`aoe` The built-in `aoe` resource is a list of AOE\$s available on a vnode. Case-sensitive. You specify the list of AOE\$s that can be requested on a vnode by setting the value of `resources_available.aoe` to that list. Each job can request at most one AOE.

Automatically added to the “resources” line in `PBS_HOME/sched_priv/sched_config`.

Cannot be modified while a vnode is provisioning.

Non-consumable. Cannot be set on the Server’s host. Can be set only by a Manager.

Format: `string_array`.
Default: unset.
Python attribute value type: str

7.4.6.2 Vnode Attributes

**current_aoe**
The `current_aoe` vnode attribute shows which AOE is currently instantiated on a vnode. Case-sensitive.

At startup, each vnode’s `current_aoe` attribute is unset. You must set the attribute to the currently instantiated AOE if you want the scheduler to be able to choose vnodes efficiently.

The value of this attribute is set automatically after a vnode is provisioned.

This attribute cannot be modified while a vnode is provisioning.

Cannot be set on the Server’s host. Settable by Manager only; visible to all.

Format: String.
Default: Unset.

**provision_enable**
This attribute controls whether the vnode can be provisioned. If set to `True`, the vnode can be provisioned.

Cannot be set on the Server’s host.

Settable by Manager only; visible to all.

Format: Boolean
Default: Unset

7.4.6.3 Server Attributes

**max_concurrent_provision**
The maximum number of vnodes allowed to be in the process of being provisioned. Settable by Manager only; readable by all. When unset, default value is used. Cannot be set to zero; previous value is retained.

Format: Integer
Default: 5
Python attribute value type: `int`
Chapter 7  Provisioning

7.4.6.4  Hook Attributes

All attributes of the provisioning hook affect provisioning. See “Hook Attributes” on page 396 of the PBS Professional Reference Guide.

7.4.6.5  Scheduler Configuration Parameters

provision_policy

Specifies the provisioning policy to be used. Valid values: avoid_provision, aggressive_provision.

avoid_provision
PBS first tries to satisfy the job’s request from free vnodes that already have the requested AOE instantiated. PBS uses node_sort_key to sort these vnodes.

If it cannot satisfy the job’s request using vnodes that already have the requested AOE instantiated, it does the following:

PBS uses node_sort_key to select the free vnodes that must be provisioned in order to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them.

Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

aggressive_provision
PBS selects vnodes to be provisioned without considering which AOE is currently instantiated.

PBS uses node_sort_key to select the vnodes on which to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them. Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

Default: “aggressive_provision”.

608  PBS  Professional 12.2 Administrator’s Guide
7.5 Configuring Provisioning

7.5.1 Overview of Configuring Provisioning

The administrator configures provisioning attributes, provides a provisioning tool, and writes a provisioning hook. The administrator configures each vnode to be provisioned with a list of AOE resources, where each resource is an AOE that is available to be run on that vnode. These resources are tags that tell the scheduler what can be run on that vnode. The administrator should also inform the scheduler about the current environment on each vnode, by setting the vnode’s `current_aoe` attribute. It is also necessary to enable provisioning on each vnode to be provisioned and to set provisioning policy at the Server and Scheduler.

7.5.1.1 Steps in Configuring Provisioning

These are the steps that the administrator must take:

1. Provide a provisioning tool such as SGI Tempo. See section 7.5.2, “Provide a Provisioning Tool”, on page 610.

2. Prepare each OS, application, or script that is to be used in provisioning. See section 7.5.3, “Prepare Images”, on page 610.

3. Configure each vnode to be provisioned with the appropriate resources. See section 7.5.4, “Define aoe Resources”, on page 610.

4. Optional: publish each vnode’s current AOE. See section 7.5.5, “Inform Scheduler of Current AOE”, on page 611.

5. Write the provisioning hook’s script. See section 7.5.6, “Write the Provisioning Script”, on page 611.

6. Create the empty provisioning hook, import the script, and configure the hook. See section 7.5.7, “Create and Configure the Provisioning Hook”, on page 613.

7. Configure provisioning policy. See section 7.5.8, “Configure Provisioning Policy”, on page 615.

8. Enable provisioning on vnodes. See section 7.5.9, “Enable Provisioning on Vnodes”, on page 616.

7.5.2 Provide a Provisioning Tool

For each vnode you wish to provision, there must be a provisioning tool that can be used on that vnode. This provisioning tool can either be written into the provisioning hook script, or be a separate script that is called by the provisioning hook script. You can write the provisioning tool yourself, or you can use something like the SGI Tempo cluster management tool. Your provisioning tool may be able to employ network-accessible power control units.

7.5.3 Prepare Images

You must prepare each image, application, or script you will use. Make sure that each is available to the target vnode. For example, if you use a diskless node server, put your images on the diskless node server.

The values for the ncpus and mem resources must be the same for all OS images that may be instantiated on a given vnode.

7.5.4 Define aoe Resources

The aoe resource is of type string_array, and is used to hold the names of the AOE(s) available at each vnode. This resource is not consumable. This resource is unset by default, and by default is added to the resources line in PBS_HOME/sched_priv/sched_config. See “Built-in Resources” on page 299 of the PBS Professional Reference Guide. The aoe resource is visible to all, but settable by the PBS Manager and Operator only.

The scheduler must be able to find out which AOE(s) can be run on which vnodes. To tag each vnode with the AOE(s) that can run on it, set that vnode’s resources_available.aoe attribute to the list of available AOE(s). For example, if vnode V1 is to run RHEL and SLES, and the hook script will recognize rhel and sles, set the vnode’s resources_available.aoe attribute to show this:

```
Qmgr: set node V1 resources_available.aoe = “rhel, sles”
```

It is recommended that you make a list of all of the AOE(s) that may be used in provisioning in your PBS complex. The list is to facilitate script writing and resource configuration. Each entry in this list should contain at least the following information:

- Full description of the AOE
- Resource name of the AOE
- Vnodes that are to run the AOE
- Location where script should look for the AOE
For example, the list might look like the following table:

Table 7-4: Example AOE List

<table>
<thead>
<tr>
<th>Description</th>
<th>Resource Name</th>
<th>Vnodes</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuSE SLES 9 64-bit</td>
<td>sles9</td>
<td>mars, jupiter, neptune, pluto</td>
<td>imageserver.example.com:/images/sles9-image</td>
</tr>
<tr>
<td>SuSE SLES 10 64-bit</td>
<td>sles10</td>
<td>mars, jupiter, pluto</td>
<td>imageserver.example.com:/images/sles10-image</td>
</tr>
<tr>
<td>Red Hat Enterprise Linux 5 32-bit</td>
<td>rhel5</td>
<td>luna, aitne, io</td>
<td>imageserver.example.com:/images/rhel5-image</td>
</tr>
<tr>
<td>Windows Server 2008 32-bit</td>
<td>winsrv8</td>
<td>luna, aitne, io</td>
<td>\WinServer\C:\images\winsrv8</td>
</tr>
</tbody>
</table>

7.5.5 Inform Scheduler of Current AOE

Each vnode has an attribute called current_aoe which is used to tell the scheduler what the vnode’s current AOE is. This attribute is unset by default. The attribute is of type string. It is visible to all, but settable by the PBS Manager only.

You can set this attribute on each vnode that will be used in provisioning. Set it to the value of the AOE that is currently instantiated on the vnode. So for example, using the table in section 7.5.4, “Define aoe Resources”, on page 610, if vnode pluto is running 64-bit SuSE SLES 10, set current_aoe to sles10:

Qmgr: set node pluto current_aoe = sles10

When PBS provisions a vnode with a new AOE, the PBS Server sets the value of current_aoe to the new AOE.

If PBS cannot provision a vnode with the desired AOE, it marks the vnode offline and unsets the value of current_aoe.

7.5.6 Write the Provisioning Script

You create the provisioning hook using a provisioning script which must manage all provisioning, either directly, or indirectly by calling other scripts. The script in the hook is the master provisioning script.
Chapter 7  Provisioning

The script that does the provisioning must have the logic needed to provision the specified vnode with the specified AOE.

There are two types of provisioning. One is when the vnode is rebooted after installing/uninstalling the OS/application or running the script. The other is when the vnode is not rebooted after installing/uninstalling the OS/application or running the script.

The master provisioning script must meet the following requirements:

- Written in Python
- Arguments to the script are the vnode name and the AOE name
- If the vnode must be rebooted for provisioning, the provisioning script must cause the target vnode to be rebooted
- Must indicate success using the correct return value:
  - Return `pbs.event.accept(0)` if provisioning is successful and the vnode is rebooted
  - Return `pbs.event.accept(1)` if provisioning is successful and the vnode is not rebooted
- Must indicate failure to PBS by using `pbs.event.reject()`
- If the master provisioning script calls other scripts, it must wait for them to finish before returning success or failure to PBS

7.5.6.1  Arguments to Master Script

The arguments to the master script are the following:

- Name of vnode to be provisioned
  Supplied to the hook via the PBS provision event object, as `pbs.event.vnode`
- Name of AOE to be instantiated on the target vnode
  Supplied to the hook via the PBS provision event object, as `pbs.event.aoe`

These values can be passed to scripts that are called by the master script.

7.5.6.2  Return Values

The master script must indicate to PBS whether it succeeded or failed in a way that PBS can understand.
7.5.6.2.i   Success

By default, `pbs.event.accept()` returns zero. The script must return different values for successful provisioning, depending on whether the vnode is rebooted:

- If provisioning is successful and the vnode is rebooted, the script must return 0 (zero) to PBS via `pbs.event.accept(0)`.
- If provisioning is successful and the vnode is not rebooted, the script must return 1 (one) to PBS via `pbs.event.accept(1)`.

7.5.6.2.ii  Failure

If provisioning fails, the script must use `pbs.event.reject()` to indicate failure. By default, `pbs.event.reject()` returns 255. To return another failure code, use the following:

```
pbs.event.reject(error message, error code)
```

where error code is any number between 2 and 255. Returning an error code in `pbs.event.reject()` is optional.

7.5.6.3   Master Script Calls Subscript

Often, the master script (the hook script) calls another script, depending on the provisioning required. The subscript does the actual provisioning of the target vnode with the requested AOE. In this case, the master script must wait for the subscript to return and indicate success or failure. The master script then propagates the result to PBS.

Example of a fragment of a master script calling a subscript:

```
return_value = os.system("/var/vendor/vendor_prov.sh " <arguments to vendor_prov.sh>)
```

7.5.7   Create and Configure the Provisioning Hook

The provisioning hook causes any provisioning to happen. The provisioning hook is a Python script which either does the provisioning directly or calls other scripts or tools. Typically the provisioning hook calls other scripts, which do the actual work of provisioning. For complete information on writing hooks, see Chapter 6, "Hooks", on page 413.

You can have at most one provisioning hook. Do not attempt to create more than one provisioning hook.

In the steps that follow, we use as examples a provisioning hook named “Provision_Hook”, and an ASCII script named “master_provision.py”. 

7.5.7.1 Create the Hook

To create the provisioning hook:

Qmgr: create hook <hook name>

For example, to create a provisioning hook called Provision_Hook:

Qmgr: create hook Provision_Hook

7.5.7.2 Import the Hook Script

If the hook script is called “master_provision.py”, and it is ASCII, and it is located in /root/data/, importing the hook script looks like this:

Qmgr: import hook Provision_Hook application/x-python default /root/data/master_provision.py

See section 6.10.5, “Importing Hooks”, on page 443 for more about importing hooks.

7.5.7.3 Configure the Hook Script

7.5.7.3.i Set Event Type

The event type for the provisioning hook is called “provision”. To set the event type:

Qmgr: set hook Provision_Hook event = provision

Do not try to assign more than one event type to the provisioning hook.

7.5.7.3.ii Set Alarm Time

The default alarm time for hooks is 30 seconds. This may be too short for a provisioning hook. You should set the alarm time to a value that is slightly more than the longest time required for provisioning. Test provisioning each AOE, and find the longest time required, then add a small amount of extra time. To set the alarm time:

Qmgr: set hook Provision_Hook alarm = <number of seconds required>
7.5.8 Configure Provisioning Policy

7.5.8.1 Set Maximum Number of Concurrently Provisioning Vnodes

The value of the Server’s max_concurrent_provision attribute specifies the largest number of vnodes that can be in the process of provisioning at any time. The default value of this attribute is 5. Set the value of this attribute to the largest number of vnodes you wish to have concurrently provisioning. See section 7.4.6.3, “Server Attributes”, on page 607 for more information on the attribute.

7.5.8.1.i Considerations

You may wish to limit the number of hosts that can be in the process of provisioning at the same time:

- So that the network isn’t overwhelmed transferring OS images to execution nodes
- So the hosts won’t draw excessive power when powering up at the same time

Many sites have tools that notify them when an execution node goes down. You may want to avoid being paged every time an execution node is provisioned with a new AOE.

7.5.8.2 Set Scheduling Policy

When a job is scheduled to be run, and the job requests an AOE, PBS can either try to fit the job on vnodes that already have that AOE instantiated, or it can choose the vnodes regardless of AOE. Choosing regardless of AOE is the default behavior; the assumption is that the chances of finding free vnodes that match all the requirements including that of the requested AOE are not very high.

Provisioning policy is controlled by the provision_policy scheduling parameter in PBS_HOME/sched_priv/sched_config. This parameter is a string which can take one of two values: avoid_provision or aggressive_provision. If you want PBS to try first to use vnodes whose AOE already match the requested AOE, set provision_policy to avoid_provision. If you want PBS to choose vnodes regardless of instantiated AOE, set it to aggressive_provision.

For details about the provision_policy parameter, see section 7.4.6.3, “Server Attributes”, on page 607.

For jobs that do not request an AOE, node_sort_key is used to choose vnodes.
7.5.9  Enable Provisioning on Vnodes

PBS will provision only those vnodes that have provisioning enabled. Provisioning on each vnode is controlled by its provision_enable attribute. This attribute is Boolean, with a default value of False. You enable provisioning on a vnode by setting its provision_enable attribute to True.

This attribute cannot be set to True on the Server’s host.

See section 7.4.6.2, “Vnode Attributes”, on page 607 for details about the provision_enable vnode attribute.

7.5.10  Enable Provisioning Hook

The last step in configuring provisioning is enabling the provisioning hook. The provisioning hook is enabled when its enabled attribute is set to True. To set the enabled attribute to True for the provisioning hook named Provision_Hook:

```
Qmgr: set hook Provision_Hook enabled = True
```

7.6  Viewing Provisioning Information

7.6.1  Viewing Provisioning Hook Contents

To see the contents of the provisioning hook, export them:

```
Qmgr: export hook <hook name> application/x-python default <output-path>/
<output-filename>
```

For example, if the provisioning hook is named Provision_Hook, and you wish to export the contents to /usr/user1/hook_contents:

```
Qmgr: export hook Provision_Hook application/x-python default /usr/
user1/hook_contents
```

7.6.2  Viewing Provisioning Hook Attributes

To view the provisioning hook’s attributes, use the list hook option to the qmgr command:

```
qmgr -c "list hook <hook name>"
```
7.6.3 Printing Provisioning Hook Creation Commands

To print the provisioning hook’s creation commands, use the `print hook` option to the `qmgr` command:

```
qmgr -c "p hook"
#
# Create hooks and set their properties.
#
#
# Create and define hook my_prov_hook
#
create hook my_prov_hook
set hook my_prov_hook type = site
set hook my_prov_hook enabled = True
set hook my_prov_hook event = provision
set hook my_prov_hook user = pbsadmin
set hook my_prov_hook alarm = 30
set hook my_prov_hook order = 1
import hook my_prov_hook application/x-python base64 -c2xzGwK
```

7.6.4 Viewing Attributes and Resources Affecting Provisioning

7.6.4.1 Server Attributes

To see the Server attributes affecting provisioning, print the Server’s information using the `qmgr` command:

```
qmgr -c "print server"
```
You will see output similar to the following:

```
# qmgr
Max open servers: 49
Qmgr: p s
#
# Create queues and set their attributes.
#
# Create and define queue workq
#
create queue workq
# set queue workq queue_type = Execution
set queue workq enabled = True
set queue workq started = True
#
# Set server attributes.
#
set server scheduling = True
set server default_queue = workq
set server log_events = 511
set server mail_from = adm
set server resv_enable = True
set server node_fail_requeue = 310
set server pbs_license_min = 0
set server pbs_license_max = 2147483647
set server pbs_license_linger_time = 31536000
set server license_count = "Avail_Global:0 Avail_Local:256 Used:0 High_Use:0"

set server max_concurrent_provision = 5
```
7.6.4.2 Viewing Vnode Attributes and Resources

To see vnode attributes and resources affecting provisioning, use the -a option to the pbsnodes command:

```bash
pbsnodes -a
host1
  Mom = host1
  ntype = PBS
  state = free
  pcpus = 2
  resources_available.aoe = osimage1, osimage2
  resources_available.arch = linux
  resources_available.host = host1
  resources_available.mem = 2056160kb
  resources_available.ncpus = 2
  resources_available.vnode = host1
  resources_assigned.mem = 0kb
  resources_assigned.ncpus = 0
  resources_assigned.vmem = 0kb
  resv_enable = True
  sharing = default_shared
  provision_enable = True
  current_aoe = osimage2
```

7.7 Requirements and Restrictions

7.7.1 Site Requirements

7.7.1.1 Single-vnode Hosts Only

PBS will provision only single-vnode hosts. Do not attempt to use provisioning on hosts that have more than one vnode.
Chapter 7  Provisioning

7.7.1.2  Provisioning Tool Required

For each vnode you wish to provision, there must be a provisioning tool that can be used on that vnode. Examples of provisioning tools are the following:

- The SGI Tempo cluster management tool
- Dual boot system
- Network-accessible power control units

7.7.1.3  Single Provisioning Hook Allowed

The PBS Server allows only one provisioning hook. If you have an existing provisioning hook and you import a provisioning script, that script will become the contents of the hook, whether or not the hook already has a script. The new script will overwrite the existing provisioning hook script.

7.7.1.4  Provisioning Hook Cannot Have Multiple Event Types

The provisioning hook cannot have more than one event type.

7.7.1.5  AOE Names Consistent Across Complex

Make AOE names consistent across the complex. The same AOE should have the same name everywhere.

7.7.2  Usage Requirements

7.7.2.1  Restriction on Concurrent AOE on Vnode

Only one AOE can be instantiated at a time on a vnode.

Only one kind of aoe resource can be requested in a job. For example, an acceptable job could make the following request:

```
-l select=1:ncpus=1:aoe=suse+1:ncpus=2:aoe=suse
```
7.7.2.2 Vnode Job Restrictions

A vnode with any of the following jobs will not be selected for provisioning:

- One or more running jobs
- A suspended job
- A job being backfilled around

7.7.2.3 Vnode Reservation Restrictions

A vnode will not be selected for provisioning for job MyJob if the vnode has a confirmed reservation, and the start time of the reservation is before job MyJob will end.

A vnode will not be selected for provisioning for a job in reservation R1 if the vnode has a confirmed reservation R2, and an occurrence of R1 and an occurrence of R2 overlap in time and share a vnode for which different AOE{s} are requested by the two occurrences.

7.7.2.4 Hook Script and AOE Must Be Compatible

The requested AOE must be available to the vnode to be provisioned. The following must be True:

- The AOE must be in the list of available AOE{s} for the vnode
- Each AOE listed on a vnode must be recognized by the provisioning hook script.
- The vnode must have provisioning enabled

7.7.2.5 Provisioning Hook Must Be Ready

- The provisioning hook must obey the following rules:
  - It must exist
  - It must have a Python script imported
  - It must be enabled
  - It must be designed to invoke an external script or command for AOE{s} that are to be used

7.7.2.6 Server Host Cannot Be Provisioned

The Server host cannot be provisioned: a MoM can run on the Server host, but that MoM’s vnode cannot be provisioned. The provision_enable vnode attribute, resources_available.aoe, and current_aoe cannot be set on the Server host.
7.7.2.7  PBS Attributes Not Available to Provisioning Hook

The provisioning hook cannot operate on PBS attributes except for the following:

- The name of the vnode to be provisioned: `pbs.event.vnode`
- The AOE to be instantiated: `pbs.event.aoe`

7.7.2.8  `avoid_provision` Incompatible with `smp_cluster_dist`

The `avoid_provision` provisioning policy is incompatible with the `smp_cluster_dist` scheduling scheduler configuration parameter. If a job requests an AOE, the `avoid_provision` policy overrides the behavior of `smp_cluster_dist`.

7.8  Defaults and Backward Compatibility

By default, PBS does not provide provisioning. You must configure PBS to provide provisioning.

7.9  Example Scripts

7.9.1  Sample Master Provisioning Hook Script With Explanation

We show a sample provisioning hook script, and an explanation of what the script does. For readability, the sample script is a master script calling two subscripts.

This provisioning hook allows two kinds of provisioning request:

- For the application AOE named “App1”, via the script `app_prov.sh`
  The `app_prov.sh` script does not reboot the vnode
- For other provisioning, via the vendor-provided provisioning shell script `vendor-prov.sh`
  The `vendor-prov.sh` script reboots the vnode
7.9.1.1 Sample Master Provisioning Hook Script

```python
import pbs                                      (1)
import os                                      (2)

e = pbs.event()                                (3)
vnode = e.vnode                                 (4)
aoe = e.aoe                                    (5)

if (aoe == "App1"):                           (6)
    appret = os.system("/var/user/app_prov.sh
                     " + vnode + " " + aoe )         (7)
    if appret != 1:                             (8)
        e.reject("Provisioning without reboot
                    failed", 210)                  (9)
    else:
        e.accept(1)                             (10)

ret = os.system("/var/vendor/vendorprov.sh
                " + vnode + " " + aoe )           (11)

if ret != 0:                                  (12)
    e.reject("Provisioning with reboot
              failed", 211)                (13)
else:
    e.accept(0)                                (14)
```

7.9.1.2 Explanation of Sample Provisioning Hook Script

- Lines 1 and 2 import the pbs and os modules.
- Line 3 puts the PBS provisioning event into the local variable named “e”.
- Lines 4 and 5 store the target vnode name and the name of the AOE to be instantiated on the target vnode in local variables.
- Line 6 checks whether provisioning of the application AOE named “App1” is requested.
- Line 7 is where the actual code to do non-rebooting provisioning could go. In this example, we call a subscript, passing the name of the target vnode and the requested AOE, and
The non-rebooting provisioning subscript should return 1 on success.

- Line 8 checks whether non-rebooting provisioning via `app_prov.sh` succeeded.
- Line 9 returns the error code 210 and an error message to PBS if `app_prov.sh` failed.
- Line 10 returns 1 via `pbs.event.accept(1)` if non-rebooting provisioning succeeded.
- Line 11 calls the vendor-supplied script that is responsible for doing rebooting provisioning whenever “App1” is not the AOE.

The name of the target vnode and the requested AOE are passed to this script. The vendor-supplied script should expect these two arguments. The return value from this script is stored in the variable named “ret”.

- Line 12 checks whether rebooting provisioning via the vendor-supplied script `vendor-prov.sh` was successful.
- Line 13: If the return value is anything but zero (success), the provisioning hook script passes the error code 211 back to PBS, along with an error message.
- Line 14 returns success to PBS via `pbs.event.accept(0)` and the master script exits.
The following is a master provisioning hook script that calls SGI Tempo:

```python
# -*- coding: utf-8 -*-
import pbs
import os

e = pbs.event()
vnode = e.vnode
aoe = e.aoe

if (aoe=="App1"):
    ret = os.system("/root/osprov/application.sh " + vnode + " " + aoe)
    if ret != 0:
        e.reject("Non-reboot provisioning failed",ret)
    else:
        e.accept(1)

ret = os.system("/root/osprov/sgi_provision.sh " + vnode + " " + aoe)
if ret != 0:
    e.reject("Reboot provisioning failed",ret)
else:
    e.accept(0)
```
Here is the provisioning script named `sgi_provision.sh`:

```bash
#!/bin/sh

scriptdir=`dirname $0`
pid=$$
ERR=255

test -f /etc/profile.d/mgr.sh && . /etc/profile.d/mgr.sh
if [ $? -ne 0 ]; then
test -f /etc/profile.d/cmgr.sh && . /etc/profile.d/cmgr.sh
fi

# do `rpm -qa |grep agnostic` to check version of installed agnostic
CMAGNOSTIC_PATH_CMIMG="/usr/sbin"
CMAGNOSTIC_PATH_CMNDE="/usr/bin"

if [ ! -x "${CMAGNOSTIC_PATH_CMIMG}/cmimage" ]; then
myecho "could not find executable ${CMAGNOSTIC_PATH_CMIMG}/cmimage"
exit $ERR
fi
if [ ! -x "${CMAGNOSTIC_PATH_CMNDE}/cmnodes" ]; then
myecho "could not find executable ${CMAGNOSTIC_PATH_CMNDE}/cmnodes"
exit $ERR
fi

function myecho
{
  echo -e "$pid: $*
}

if [ $# -lt 2 ]; then
myecho "syntax: $0 <machine> <aoe name>"
exit $ERR
fi

myecho "starting: $0 $*"
```
machine=$1
aoe_name=$2

hostname='hostname'
if [ "$machine" = "$hostname" ]; then
  myecho "Cannot reboot own machine. Please provide another machine name"
  exit $ERR
fi

# check hostname in the list of compute nodes
vnodes=`${CMAGNOSTIC_PATH_CMNDE}/cmnodes --list --group compute`
if [ $? -ne 0 ]; then
  myecho "${CMAGNOSTIC_PATH_CMNDE}/cmnodes returned error"
  exit $ERR
fi

  echo -e "$vnodes" | grep "^${machine}$"
  if [ $? -ne 0 ]; then
    myecho "Vnode $machine is not known to me"
    exit $ERR
  fi

# check aoe name in list of cmimages available
aoes=`${CMAGNOSTIC_PATH_CMIMG}/cmimage --images | awk '{print $1}' | awk
  "image=" '{print $2}'`
if [ $? -ne 0 ]; then
  myecho "${CMAGNOSTIC_PATH_CMIMG}/cmimage returned error"
  exit $ERR
fi

  echo -e "$aoes" | grep "^${aoe_name}$"
  if [ $? -ne 0 ]; then
    myecho "AOE/image $aoe_name is not known to me"
    exit $ERR
  fi
curr_image=`${CMAGNOSTIC_PATH_CMIMG}/cmimage --running --node "$machine" | awk -F"image=" '{print $2}' | awk '{print $1}'`
if [ "$cur_image" = "$aoe_name" ]; then
myecho "AOE/image $aoe_name already running on $machine"
exit 0
fi

#okay all set now, kick off provisioning
${CMAGNOSTIC_PATH_CMIMG}/cmimage --set --node ${machine} --image ${aoe_name}
if [ $? -ne 0 ]; then
myecho "setting of image $aoe_name failed"
exit $ERR
fi

${CMAGNOSTIC_PATH_CMIMG}/cmimage --reboot --node ${machine}
if [ $? -ne 0 ]; then
myecho "rebooting of node $machine failed"
exit $ERR
fi

curr_image=`${CMAGNOSTIC_PATH_CMIMG}/cmimage --running --node "$machine" | awk -F"image=" '{print $2}' | awk '{print $1}'`
if [ "$?" -ne 0 ]; then
myecho "checking of machine $machine failed"
exit $ERR
fi
if [ "$curr_image" != "$aoe_name" ]; then
myecho "AOE/image $aoe_name could not be set on $machine"
exit $ERR
fi
myecho "$machine is now up with AOE $aoe_name"
exit 0

7.9.3 Sample Script Set

This is a set of example Linux scripts designed to work together. They are the following:

provision_hook.py
This is the script for the provisioning hook. It calls the master provisioning script.

provision_master.py:
This is the master provisioning script. It is responsible for rebooting the machine being provisioned. It calls update_grub.sh to update the current AOE.

update_grub.sh
This shell script updates the linux grub.conf file and sets the value for current_aoe after the reboot.

The update_grub.sh script must be modified according to the grub configuration of the system in question before being run.
Chapter 7  Provisioning

7.9.3.1  Provisioning Hook Script

provision_hook.py:

```python
import pbs
import os

e = pbs.event()
vnode = e.vnode
aoe = e.aoe

#print "vnode:" + vnode
#print "AOE:" + aoe

if (aoe=="App1"):
    print "Provisioning an application"
e.accept(1)

ret = os.system("python /root/provision_master.py " + vnode + " " + aoe + " " + "lin")
#print "Python top level script returned " + str(ret)
if ret != 0:
e.reject("Provisioning failed",ret)
else:
e.accept(0)
```
7.9.3.2 Master Provisioning Script

provision_master.py:

```
#!/usr/bin/python

#---------------------
# success : 0
# failure : 1
#---------------------
# win_or_lin == 1 : windows
# win_or_lin == 0 : linux
#---------------------
# 1 is TRUE
# 0 is FALSE
#---------------------

import sys
import os

vnode = sys.argv[1]
aoe = sys.argv[2]
win_or_lin = sys.argv[3]

print vnode, aoe

if not aoe.find('win'):
    print "aoe is win"
    isvnodewin = 1
else:
    print "aoe is *nix"
    isvnodewin = 0

print "win_or_lin = [", win_or_lin, "]"

if (win_or_lin == "win"):
    print "entering window server"
```
if isvnodewin:
    #------------ WINDOWS -> WINDOWS
    ret = os.system("pbs-sleep 05")
    #------------ WINDOWS -> WINDOWS

else:
    #------------ WINDOWS -> LINUX
    ret = os.system("pbs-sleep 05")
    #------------ WINDOWS -> LINUX

ret = os.system("pbs-sleep 45")
print "Pinging machine until it is up..."
timeout = 120
ticks = 0

while 1:
    ret = os.system("ping -c 1 -i 5 " + vnode + " -w 10 > /dev/null 2>&1")
    if not ret:
        print "that machine is now up"
        exit(0)

    ticks = ticks + 1
    print "ticks = ", ticks
    if ticks > timeout:
        print "exit ticks = ", ticks
        print "that machine didn't come up after 2 mins,FAIL"
        exit(1)

else:
    print "entering linux server"
    if isvnodewin:
        #------------ LINUX -> WINDOWS
        ret = os.system("sleep 05")
        #------------ LINUX -> WINDOWS
    else:

    else:
#------------ LINUX -> LINUX

ret = os.system("scp -o StrictHostKeyChecking=no /root/update_grub.sh "+ vnode + ":/root > /dev/null 2>&1")
if ret != 0:
    print "scp failed to copy"
    exit(1)

ret = os.system("/usr/bin/ssh -o StrictHostKeyChecking=no " + vnode + " "/root/update_grub.sh " + vnode + " " + aoe + " l " + " "> /dev/null 2>&1")
if ret != 0:
    print "failed to run script"
    exit(1)

ret = os.system("/usr/bin/ssh -o StrictHostKeyChecking=no " + vnode + " \"reboot\" " + " > /dev/null 2>&1")
if ret != 0:
    print "failed to reboot that machine"
    exit(1)

#------------ LINUX -> LINUX

ret = os.system("sleep 45")
print "Pinging machine until it is up..."
timeout = 120
ticks = 0

while 1:
    ret = os.system("ping -c 1 -i 5 " + vnode + " -w 10 > /dev/null 2>&1")
    if not ret:
        print "that machine is now up"
        exit(0)

    print "ticks = ", ticks
    ticks = ticks + 1
if ticks > timeout:
    print "That machine didn't come up after 2 mins. FAIL"
    exit(1)
7.9.3.3 Grub Update Shell Script

update_grub.sh:

```bash
#!/bin/sh

if [ $# -lt 2 ]; then
    echo "syntax: $0 <machine ip> <aoe name>
    exit 1
fi

machine=$1
aoe_name=$2

menufile="/boot/grub/grub.conf"
if [ ! -f "$menufile" ]; then
    echo "grub.conf file not found. $machine using grub bootloader?"
    exit 1
fi

link=`ls -l $menufile | cut -c1`
if [ "$link" = "l" ]; then
    menufile=`ls -l $menufile | awk -F"-> " '{print $2}'`
    echo "Found link file, original file is $menufile"
fi

titles=`cat $menufile | grep title | awk -F"title " '{print $2}' | sed 's/\t/\t/g'`
lines=`echo -e "$titles" | wc -l`

found_aoe_index=-1
count=0
while [ $count -lt $lines ]
do
    lineno=`expr $count + 1`
    title=`echo -e "$titles" | head -n $lineno | tail -n 1`
    if [ "$aoe_name" = "$title" ]; then
```
found_aoe_index=$count
fi
count=`expr $count + 1`
done

if [ $found_aoe_index = -1 ]; then
echo "Requested AOE $aoe_name is not found on machine $machine"
exit 2
fi

new_def_line="default=$found_aoe_index"
def_line=`cat $menufile | grep "^default="`

echo "new_def_line=$new_def_line"
echo "def_line=$def_line"
echo "menufile=$menufile"

cp $menufile /boot/grub/grub.conf.backup
cat $menufile | sed "s/"$def_line/$new_def_line/g" > grub.out
if [ -s grub.out ]; then
mv grub.out $menufile
else
exit 1
fi

service pbs stop

exit 0
7.10 Advice and Caveats

7.10.1 Using Provisioning Wisely

It is recommended that when using provisioning, you set PBS up so as to prevent things such as the following:

• User jobs not running because vnodes used in a reservation have been provisioned, and provisioning for the reservation job will take too long
• Excessive amounts of time being taken up by provisioning from one AOE to another and back again

In order to avoid problems like the above, you can do the following to keep specific AOE requests together:

• For each AOE, associate a set of vnodes with a queue. Use a hook to move jobs into the right queues.
• Create a reservation requesting each AOE, then use a hook to move jobs requesting AOE into the correct reservation.
7.10.1.1 Preventing Provisioning

You may need to prevent specific users or groups from using provisioning. You can use a job submission, job modification, or reservation creation hook to prevent provisioning. For more about hooks, see Chapter 6, "Hooks", on page 413. The following is an example of a hook script to prevent USER1 from provisioning:

```python
import pbs
import re

#--- deny user access to provisioning

e = pbs.event()
j = e.job   #--- Use e.resv to restrict provisioning in reservation
who = e.requestor

unallow_u_list = ["USER1"]

if who not in unallow_u_list
    e.accept(0)

#User request AOE in select?
if j.Resource_List["select"] != None:
s = repr(j.Resource_List["select"])
if re.search("aoe=", s) != None:
    pbs.logmsg(pbs.LOG_DEBUG, "User %s not allowed to provision" % (who))
    e.reject("User not allowed to provision")

#User request AOE?
if j.Resource_List["aoe"] != None:
    pbs.logmsg(pbs.LOG_DEBUG, "User %s not allowed to provision" % (who))
    e.reject("User not allowed to provision")

e.accept(0)
```
7.10.2 Allow Enough Time in Reservations

If a job is submitted to a reservation with a duration close to the walltime of the job, provisioning could cause the job to be terminated before it finishes running, or to be prevented from starting. If a reservation is designed to take jobs requesting an AOE, leave enough extra time in the reservation for provisioning.

7.11 Errors and Logging

7.11.1 Errors

A vnode is marked offline if:

- Provisioning fails for the vnode
- The AOE reported by the vnode does not match the requested AOE after the provisioning script finishes

A vnode is not marked offline if provisioning fails to start due to internal errors in the script.

7.11.2 Logging

7.11.2.1 Accounting Logs

For each job and reservation, an accounting log entry is made whenever provisioning starts and provisioning ends. Each such log entry contains a list of the vnodes that were provisioned, the AOE that was provisioned on these vnodes, and the start and end time of provisioning.

The accounting log entry for the start of provisioning is identified by the header “P”, and the entry for the end of provisioning is identified by the header “p”.

Example:

Printed when job starts provisioning:

```
01/15/2009 12:34:15;P;108.mars;user=user1 group=group1 jobname=STDIN queue=workq prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=START start_time=1231928746
```

Printed when job stops provisioning:

```
01/15/2009 12:34:15;p;108.mars;user=user1 group=group1 jobname=STDIN queue=workq prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=END status=SUCCESS end_time=1231928812
```
Printed when provisioning for job failed:

“01/15/2009 12:34:15;p;108.mars:user=user1 group=group1 jobname=STDIN
queue=workq prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1
provision_event=END status=FAILURE end_time=1231928812”

7.1.2.2 Server Logs

7.1.2.2.i Messages Printed at Log Event Class 0x0080

“vnode <vnode name>: Vnode offlined since it failed provisioning”
“vnode <vnode name>: Vnode offlined since server went down during provisioning”
“Provisioning for Job <job id> succeeded, running job”
“Job failed to start provisioning”
“Provisioning for Job <job id> failed, job held”
“Provisioning for Job <job id> failed, job queued”

7.1.2.2.ii Messages Printed at Log Event Class 0x0100

“Provisioning of Vnode <vnode name> successful”
“Provisioning of <vnode name> with <AOE name> for <job ID> failed, provisioning exit status=<number>”
“Provisioning of <vnode name> with <aoe name> for <job id> timed out”
“Provisioning vnode <vnode> with AOE <AOE> started successfully”
“provisioning error: AOE mis-match”
“provisioning error: vnode offline”

7.1.2.2.iii Messages Printed at Log Event Class 0x0002

“Provisioning hook not found”

7.1.2.2.iv Messages Printed at Log Event Class 0x0001

“Provisioning script recompilation failed”

7.1.2.3 Scheduler Logs

7.1.2.3.i Messages Printed at Log Event Class 0x0400

Printed when vnode cannot be selected for provisioning because requested AOE is not available on vnode:

“Cannot provision, requested AOE <aoe-name> not available on vnode”
Printed when vnode cannot be selected for provisioning because vnode has running or suspended jobs, or the reservation or job would conflict with an existing reservation:

“Provision conflict with existing job/reservation”

Printed when vnode cannot be selected for provisioning because provision_enable is unset or set False on vnode:

“Cannot provision, provisioning disabled on vnode”

Printed when job cannot run because server is not configured for provisioning:

“Cannot provision, provisioning disabled on server”

Printed when multiple vnodes are running on the host:

“Cannot provision, host has multiple vnodes”

Printed when vnodes are sorted according to avoid_provision policy:

“Re-sorted the nodes on aoe <aoe name>, since aoe was requested”

7.11.2.3.ii Messages Printed at Log Event Class 0x0100

Printed when a vnode is selected for provisioning by a job:

“Vnode <vnode name> selected for provisioning with <AOE name>”

7.11.3 Error Messages

Printed when vnode is provisioning and current_aoe is set or unset or resources_available.aoe is modified via qmgr:

“Cannot modify attribute while vnode is provisioning”

Printed when qmgr is used to change state of vnode which is currently provisioning:

“Cannot change state of provisioning vnode”

Printed when vnode is deleted via 'qmgr > delete node <name>' while it is currently provisioning:

“Cannot delete vnode if vnode is provisioning”

Printed when provision_enable, current_aoe or resources_available.aoe are set on host running PBS server and scheduler:

“Cannot set provisioning attribute on host running PBS server and scheduler”
Chapter 7  Provisioning

Printed when current_aoe is set to an AOE name that is not listed in resources_available.aoe of the vnode:

“Current AOE does not match with resources_available.aoe”

Printed when an event of a hook is set to 'provision' and there exists another hook that has event 'provision':

“Another hook already has event 'provision', only one 'provision' hook allowed”

Printed when qsub has -laoe and -lselect=aoe:

“-lresource= cannot be used with "select" or "place", resource is: aoe”

Job comment printed when job fails to start provisioning:

“job held, provisioning failed to start”

Printed when job is submitted or altered so that it does not meet the requirements that all chunks must request same AOE, and this AOE must match that of any reservation to which the job is submitted:

“Invalid provisioning request in chunk(s)”
Security

This chapter describes the security features of PBS. These instructions are for the PBS administrator and Manager.

8.1 Configurable Features

This section gives an overview of the configurable security mechanisms provided by PBS, and gives links to information on how to configure each mechanism.

The following table lists configurable PBS security mechanisms and their configuration procedures.

Table 8-1: Security Mechanisms and their Configuration Procedures

<table>
<thead>
<tr>
<th>Security Mechanism</th>
<th>Configuration Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access control for Server, queues, reservations</td>
<td>&quot;Using Access Control” on page 647</td>
</tr>
<tr>
<td>Event logging for Server, Scheduler, MoMs</td>
<td>&quot;Event Logging” on page 867</td>
</tr>
<tr>
<td>File copy mechanism</td>
<td>&quot;Setting File Transfer Mechanism” on page 881</td>
</tr>
<tr>
<td>Levels of privilege (user roles)</td>
<td>&quot;Setting User Roles” on page 644</td>
</tr>
<tr>
<td>Restricting access to execution hosts via $restrict_user</td>
<td>&quot;Restricting Execution Host Access” on page 668</td>
</tr>
</tbody>
</table>
8.2 Setting User Roles

8.2.1 Root Privilege

Root privilege is required to perform some operations in PBS involving writing to the Server’s private, protected data. Root privilege is required in order to do the following:

- Alter MoM and Scheduler configuration files
- Set scheduler priority formula
- Run the following commands:
  - pbs_probe
  - pbs_mom
  - pbs_sched
  - pbs_server
  - pbsfs
- Use the `tracejob` command to view accounting log information

There are some operations that root privilege alone does not allow. These operations require Manager privilege but not root privilege. Manager privilege, but not root privilege, is required in order to do the following:

- Set attributes
- Create or delete vnodes using the `qmgr` command, which is the supported method.

8.2.2 User Roles

PBS allows certain privileges based on what role a person has, and whether that person has root privilege. PBS recognizes only three roles, and all those using PBS must be assigned one of these roles. These roles are `Manager`, `Operator`, and `user`. Roles are assigned by PBS Managers only. No roles can be added, and roles cannot be modified; the function of roles is hardcoded in the Server.

In addition to these roles, PBS requires a PBS administrator to perform some downloading, installation, upgrading, configuration, and management functions. PBS does not recognize `PBS administrator` as a PBS role; this term is used in PBS documentation to mean the person who performs these tasks.

PBS roles and PBS administrators are described in the following sections:
8.2.2.1 User

8.2.2.1.i Definition of User

Users are those who submit jobs to PBS.

Users have the lowest level of privilege. Users are referred to in the PBS documentation as “users”. By default, users may operate only on their own jobs. They can do the following:

- Submit jobs
- Alter, delete, and hold their own jobs
- Status their own jobs, and those of others if permission has been given via the `query_other_jobs` Server attribute. The `query_other_jobs` Server attribute controls whether unprivileged users are allowed to select or query the status of jobs owned by other users. This attribute is a Boolean, with default value of `False`, and can be set by a Manager only. See “Server Attributes” on page 316 of the PBS Professional Reference Guide.
- List and print some but not all Server, queue, vnode, Scheduler, and reservation attributes

8.2.2.1.ii Defining List of Users

PBS allows you to define a list of users allowed or denied access to the PBS Server, however this is done using the PBS access control list mechanism. Access control is described in section 8.3, “Using Access Control”, on page 647.

8.2.2 Operator

8.2.2.1 Definition of Operator

A PBS Operator is a person who has an account that has been granted Operator privilege. Operators have more privilege than users, and less privilege than Managers.

Operators can manage the non-security-related attributes of PBS such as setting and unsetting non-security attributes of vnodes, queues, and the Server. Operators can also set queue ACLs. Operators can do the following:

- All operations that users can perform
- Set non-security-related Server, queue, and vnode attributes (Operators are not permitted
to set Server ACLs)

- Alter some job attributes
- Set or alter most resources on the Server, queues, and vnodes
- Rerun, requeue, delete, and hold all jobs
- Run any command to act on a job

8.2.2.2.ii Defining List of Operators

To define the list of Operators at a PBS complex, set the Server’s `operators` attribute to a list of usernames, where each username should be an Operator. See “Server Attributes” on page 316 of the PBS Professional Reference Guide.

It is important to grant Operator privilege to appropriate persons only, since Operators can control how user jobs run.

8.2.2.3 Manager

8.2.2.3.i Definition of Manager

A Manager is a person who has an account that has been granted PBS Manager privilege. Managers have more privilege than Operators. Managers can manage the security aspects of PBS such as Server ACLs and assignment of User Roles.

Managers can do the following:

- All operations that Operators can perform
- Create or delete queues or vnodes
- Set all Server, queue, and vnode attributes, including Server ACLs

8.2.2.3.ii Defining List of Managers

To define the list of Managers at a PBS complex, set the Server’s `managers` attribute to a list of usernames, where each username should be a Manager. See “Server Attributes” on page 316 of the PBS Professional Reference Guide.

If the Server’s `managers` attribute is not set or is unset, root on the Server’s host is given Manager privilege.

It is important to grant Manager privilege to appropriate persons only, since Managers control much of PBS.
8.2.2.4 PBS Administrator

8.2.2.4.i Definition of PBS Administrator
A PBS administrator is a person who has an account with root privilege, and a separate account with Manager privilege.
PBS administrators perform all the functions requiring root privilege, as described in section 8.2.1, “Root Privilege”, on page 644.

8.2.2.4.ii Defining PBS Administrators
A PBS administrator is a person with both root privilege and Manager privilege.

8.3 Using Access Control

8.3.1 Access Definitions
In this section we describe the meaning of access for each entity and object where the access of the entity to the object has an access control mechanism.

8.3.1.1 Access to a PBS Object
Below are the definitions of what access to each of the following PBS objects means:

Access to the Server
Being able to run PBS commands to submit jobs and perform operations on them such as altering, selecting, and querying status. It also means being able to get the status of the Server and queues.

Access to a queue
Being able to submit jobs to the queue, move jobs into the queue, being able to perform operations on jobs in the queue, and being able to get the status of the queue.

Access to a reservation
Being able to place jobs in the reservation, whether by submitting jobs to the reservation or moving jobs into the reservation. It also means being able to delete the reservation, and being able to operate on the jobs in the reservation.
Access by a PBS Entity

Access can be granted at the Server, queues, and reservations for each of the following entities:

- **User access**
  The specified user is allowed access.

- **Group access**
  A user in the specified group is allowed access

- **Host access**
  A user is allowed access from the specified host

Requirement for Access

In order to have access to a PBS object such as the Server or a queue, a user must pass all enabled access control tests: the user must be allowed access, the user’s group must be allowed access, and the host where the user is working must be allowed access.

In some cases, Manager or Operator privilege overrides access controls. For some kinds of access, there are no controls. See section 8.3.1.1, “Operations Controlled by ACLs”, on page 663.

Managing Access

PBS uses access control lists (ACLs) to manage access to the Server, queues, and reservations. There is a separate set of ACLs for the Server, each queue, and each reservation. The Server enforces the access control policy for User Roles supported by PBS. The policy is hardcoded within the Server. ACLs can specify which entities are allowed access and which entities are denied access.

Each Server and queue ACL can be individually enabled or disabled by a Manager. If an ACL is enabled, access is allowed or denied based on the contents of the ACL. If the ACL is disabled, access is allowed to all. The contents of each Server or queue ACL can be set or altered by a Manager.

Reservation ACLs are enabled only by the reservation creator. The Server’s `resv_enable` attribute controls whether reservations can be created. When this attribute is set to `False`, reservations cannot be created.

No default ACLs are shipped.
8.3.4 ACLs

An ACL, or Access Control List, is a list of zero or more entities (users, groups, or hosts from which users or groups may be attempting to gain access) allowed or denied access to parts of PBS such as the Server, queues, or reservations. A Server ACL applies to access to the Server, and therefore all of PBS. A queue’s ACL applies only to that particular queue. A reservation’s ACL applies only to that particular reservation. The Server, each queue, and each reservation has its own set of ACLs.

8.3.4.1 Format of ACLs

Entity access is controlled according to the list of entities allowed or denied access as specified in the object’s acl_<entity> attribute. The object’s access control attribute contains a list of entity names, where each entity name is marked with a plus sign (“+”) if the entity is allowed access, and with a minus sign (“-”) if the entity is denied access. For example, to allow User1@host1.example.com, and deny User2@host1.example.com:

+User1@host1.example.com, -User2@host1.example.com

8.3.4.2 Default ACL Behavior

If an entity name is included without either a plus or a minus sign, it is treated as if it has a plus sign, and allowed access.

If an entity name is not in the list, the default behavior is to deny access to the entity. Therefore, if the list is empty but enabled because the object’s acl_<entity>_enable attribute is set to True (see section 8.3.5, “Enabling Access Control”, on page 652), all entities are denied access.

8.3.4.3 Modifying ACL Behavior

You can specify how an ACL treats an unmatched entity by including special flags in the ACL itself. These are the plus and minus signs.

To allow access for all unmatched entities (the reverse of the default behavior), put a plus sign (“+”) anywhere by itself in the list. For example:

+User1@host1.example.com, +, -User2@host1.example.com

To deny access for all unmatched entities (the default behavior), put a minus sign (“-”) anywhere by itself in the list. For example:

+User1@host1.example.com, -, -User2@host1.example.com

If there are entries for both a plus and a minus sign, the last entry in the list (closest to the rightmost side of the list) will control the behavior of the ACL.
8.3.4.4 Contents of User ACLs

User ACLs contain a username and hostname combination. The subject’s username and hostname combination is compared to the entries in the user ACL. Usernames take this form:

User1@host.domain.com
User1@host.subdomain.domain.com

User names can be wildcarded. See section 8.3.4.7, “Wildcards In ACLs”, on page 650.

8.3.4.5 Contents of Group ACLs

Group ACLs contain names based on the user’s default group, as defined by the operating system where the Server is executing. The subject's default group name on the Server is compared to the entries in the Group ACL. Group names cannot be wildcarded.

8.3.4.6 Contents of Host ACLs

Host ACLs contain fully-qualified hostnames. The subject's host name is compared to the entries in the host ACL. To find the fully-qualified name of a host, use the pbs_hostname command. See “pbs_hostname” on page 49 of the PBS Professional Reference Guide.

Hostnames can be wildcarded. See the following section.

8.3.4.7 Wildcards In ACLs

Usernames and hostnames can be wildcarded. The hostname portion of the username is wildcarded exactly the same way a hostname is wildcarded. The non-hostname portion of a username cannot be wildcarded.

The only character that can be used to wildcard entity names is the asterisk (“*”). Wildcarding must follow these rules:

- The asterisk must be to the right of the at sign (“@”)
- There can be at most one asterisk per entity name
- The asterisk must be the leftmost label after the at sign

The following table shows how hostnames are wildcarded:

<table>
<thead>
<tr>
<th>Wildcard Use</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.test.example.com</td>
<td>Any host in the test subdomain in example.com</td>
</tr>
</tbody>
</table>

650 PBS Professional 12.2 Administrator’s Guide
The following examples show how wildcarding works in host ACLs:

Example 8-1: To limit host access to host myhost.test.example.com only:

\texttt{myhost.test.example.com}

Example 8-2: To limit host access to any host in the test.example.com subdomain:

\texttt{*.test.example.com}

Example 8-3: To limit host access to any host in example.com:

\texttt{*.example.com}

Example 8-4: To allow host access for all hosts:

\texttt{*}

The following examples show how wildcarding works in user ACLs:

Example 8-5: To limit user access to UserA requesting from host myhost.test.example.com only:

\texttt{UserA@myhost.test.example.com}

Example 8-6: To limit user access to UserA on any host in the test.example.com subdomain:

\texttt{UserA@*.test.example.com}

Example 8-7: To limit user access to UserA on any host in example.com:

\texttt{UserA@*.example.com}

Example 8-8: To limit user access to UserA from anywhere:

\texttt{UserA@*}

\texttt{or}

\texttt{UserA}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Wildcard Use} & \textbf{Meaning} \\
\hline
\texttt{*.example.com} & Any host in example.com \\
\hline
\texttt{*.com} & Any host in .com \\
\hline
\texttt{*} & Any host \\
\hline
\end{tabular}
\caption{How Hostnames Are Wildcarded}
\end{table}
Listing a username without specifying the host or domain is the equivalent of listing the username followed by “@*”. This means that

User1

is the same as

User1@

8.3.4.8 Restrictions on ACL Contents

All access control lists are traversed from left to right, and the first match found is used. It is important to make sure that entries appear in the correct order. To single out a few, specify those few first, to the left of the other entries.

Example 8-9: To allow all users in your domain except User1 access, the list should look like this:

-User1@example.com, +*@example.com

Example 8-10: To deny access to all users in your domain except User1, the list should look like this:

+User1@example.com, -*@example.com

8.3.5 Enabling Access Control

Each Server and queue ACL is controlled by a Boolean switch whose default value is False, meaning that access control is turned off. When access control is turned off, all entities have access to the Server and to each queue. When access control is turned on, access is allowed only to those entities specifically granted access.

To use access control, first set the contents of the ACL, then enable it by setting its switch to True.

Reservation ACLs are enabled when the reservation creator sets their contents. Reservation ACLs do not have switches. See section 8.3.9.1, “Creating and Enabling Reservation Queue ACLs”, on page 662.

Reservations use queues, which are regular queues whose ACL values have been copied from the reservation. See section 8.3.9, “How Reservation Access Control Works”, on page 661.
8.3.5.1 Table of ACLs and Switches

The following table lists the ACLs and their switches, with defaults, for the Server, queues, and reservations.

<table>
<thead>
<tr>
<th>Table 8-3: ACLs and Their Switches</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User (Default Value)</strong></td>
</tr>
<tr>
<td>---------------------------</td>
</tr>
<tr>
<td><strong>Server</strong> Switch acl_user_enable (False) None acl_host_enable (False)</td>
</tr>
<tr>
<td><strong>List</strong> acl_users (all users allowed) None acl_hosts (all hosts allowed)</td>
</tr>
<tr>
<td><strong>Queue</strong> Switch acl_user_enable (False) acl_group_enable (False) acl_host_enable (False)</td>
</tr>
<tr>
<td><strong>List</strong> acl_users (all users allowed) acl_groups (all groups allowed) acl_hosts (all hosts allowed)</td>
</tr>
<tr>
<td><strong>Reservation</strong> Switch None None None</td>
</tr>
<tr>
<td><strong>List</strong> Authorized_Users (creator only) Authorized_Groups (creator’s group only) Authorized_Hosts (all hosts allowed)</td>
</tr>
<tr>
<td><strong>Reservation queue</strong> Switch acl_user_enable (True) acl_group_enable (False) acl_host_enable (False)</td>
</tr>
<tr>
<td><strong>List</strong> acl_users (creator only) acl_groups (all groups allowed) acl_hosts (all hosts allowed)</td>
</tr>
</tbody>
</table>

8.3.6 Creating and Modifying ACLs

Server and queue ACLs follow the same rules for creation and modification. Reservation queue ACLs behave the same way regular queue ACLs do. Reservation ACLs can only be created by the reservation creator, and cannot be modified. See section 8.3.8, “Reservation Access”, on page 660.
8.3.6.1 Rules for Creating and Modifying Server and Queue ACLs

• Server and queue ACLs are created and modified using the `qmgr` command.

• An ACL is a list of entries. When you operate on the list, the first match found, searching from left to right, is used. If there is more than one match for the entity you wish to control, ensure that the first match gives the behavior you want.

• When you create or add to an ACL, you can use the + or - operators to specify whether or not an entity is allowed access. Omitting the operator is equivalent to adding a + operator.

• When you re-create an existing ACL, this is equivalent to unsetting the old ACL and creating a new one.

• When you add to an ACL, the new entry is appended to the end of the ACL, on the right-hand side.

• When you remove an entity from an ACL, you cannot use + or - operators to specify which entity to remove, even if there are multiple entries for an entity and each entry has a different operator preceding it, for example “-bob, +bob”.

• When you remove an entity, only the first match found is removed.

8.3.6.2 Examples of Creating and Modifying Server and Queue ACLs

The following examples show the Server’s user ACL being set. Queue ACLs work the same way as Server ACLs, and the equivalent `qmgr` command can be used for queues. So, where we use the following for the Server:

```
Qmgr: set server acl_users ...
```

the same effect can be achieved at the queue using this:

```
Qmgr: set queue <queue name> acl_users ...
```
If the queue name is Q1, the qmgr command looks like this:

```
Qmgr: set queue Q1 acl_users ...
```

Example 8-11: To create a Server or queue ACL:

```
Qmgr: set <object> <ACL> = <entity list>
```

Example:

```
Qmgr: set server acl_users =
   "-User1@*.example.com, +User2@*.example.com"
```

ACL looks like this:

-User1@*.example.com, +User2@*.example.com

Example 8-12: To add to a Server or queue ACL:

```
Qmgr: set <object> <ACL> += <entity list>
```

Example:

```
Qmgr: set server acl_users += -User3@*.example.com
```

ACL looks like this:

-User1@*.example.com, +User2@*.example.com,
  -User3@*.example.com

Example 8-13: To remove an entry from an ACL:

```
Qmgr: set <object> <ACL> -= <entity>
```

Example:

```
Qmgr: set server acl_users -= User2@*.example.com
```

ACL looks like this:

-User1@*.example.com, -User3@*.example.com

Example 8-14: To remove two entries for the same entity from an ACL:

```
Qmgr: set <object> <ACL> -= <entity1, entity1>
```

Example: If ACL contains +A, +B, -C, -A, +D, +A

```
Qmgr: set server acl_users -= "A, A"
```

ACL looks like this:

+B, -C, +D, +A

Example 8-15: To remove multiple entities from an ACL:

```
Qmgr: set <object> <ACL> -= <entity list>
```

PBS Professional 12.2 Administrator’s Guide  655
Example: If ACL contains +B, -C, +D, +A

```
Qmgr: set server acl_users -= "B, D"
```

ACL looks like this:

- C, +A

### 8.3.6.3 Who Can Create, Modify, Enable, or Disable ACLs

The following table summarizes who can create, modify, enable, or disable ACLs and their associated switches:

<table>
<thead>
<tr>
<th>ACLs and Switches</th>
<th>Manager</th>
<th>Operator</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server ACLs and Switches</td>
<td>Create</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Modify</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Enable</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Disable</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Queue ACLs and Switches</td>
<td>Create</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Modify</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Enable</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Disable</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Reservation ACLs</td>
<td>Create</td>
<td>Only if reservation creator</td>
<td>Only if reservation creator</td>
</tr>
<tr>
<td></td>
<td>Modify</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Enable</td>
<td>Only if reservation creator</td>
<td>Only if reservation creator</td>
</tr>
<tr>
<td></td>
<td>Disable</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 8-4: Who Can Create, Modify, Enable, Disable ACLs

<table>
<thead>
<tr>
<th>ACLs and Switches</th>
<th>Manager</th>
<th>Operator</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservation Queue ACLs and Switches</td>
<td>Create</td>
<td>Yes</td>
<td>Yes, Indirectly when creating reservation</td>
</tr>
<tr>
<td></td>
<td>Modify</td>
<td>Yes</td>
<td>Yes, No</td>
</tr>
<tr>
<td></td>
<td>Enable</td>
<td>Yes</td>
<td>Yes, Indirectly when creating reservation</td>
</tr>
<tr>
<td></td>
<td>Disable</td>
<td>Yes</td>
<td>Yes, Group and host ACLs can be indirectly disabled by user during reservation creation. User ACL cannot be disabled by user.</td>
</tr>
</tbody>
</table>
8.3.6.4 Who Can Operate on Server ACLs

PBS Managers only can create or modify Server ACLs and the Boolean switches that enable them.

8.3.6.5 Who Can Operate on Queue ACLs

PBS Managers and Operators, but not users, can create and modify queue ACLs and their Boolean switches.

8.3.6.6 Who Can Operate on Reservation ACLs

When creating a reservation, the reservation creator cannot disable the user ACL, but can choose to enable or disable the group and host ACLs implicitly via the command line, and can specify the contents of all three ACLs. Reservation ACLs cannot be modified or disabled.

8.3.6.7 Who Can Operate on Reservation Queue ACLs

Unprivileged users cannot directly create, modify, enable, or disable reservation queue ACLs or the associated switches. The reservation creator can indirectly create and enable the reservation queue’s ACLs during reservation creation. If a user wants to modify a reservation queue’s ACLs, they can do so indirectly by deleting the reservation and creating a new one with the desired ACLs.

PBS Managers and Operators can modify, enable, or disable a reservation queue’s ACLs.

A reservation queue’s user ACL is always enabled unless explicitly disabled after creation by a Manager or Operator.

8.3.7 Server and Queue ACLs

Access control for an entity such as a user, group, or host is enabled by setting the attribute enabling that entity’s ACL to True. When this attribute is True, entity access is controlled according to the list of entities allowed or denied access as specified in the ACL for that entity. The default value for each ACL’s switch attribute is False, meaning that entity access is not controlled.

8.3.7.1 Server ACLs

The Server has two ACLs: a host ACL and a user ACL.
Server access is controlled by these attributes:

- User access: `acl_user_enable` and `acl_users`
- Host access: `acl_host_enable` and `acl_hosts`

### 8.3.7.2 Queue ACLs

Each queue has three ACLs: a host ACL, a user ACL, and a group ACL.

Queue access is controlled by these attributes:

- User access: `acl_user_enable` and `acl_users`
- Group access (queue only): `acl_group_enable` and `acl_groups`
- Host access: `acl_host_enable` and `acl_hosts`

### 8.3.7.3 Examples of Setting Server and Queue Access

To restrict access to the Server or queue, first set the contents of the ACL, then enable the ACL by setting its switch to `True`.

Example 8-16: To allow Server access for all users in your domain except User1, and to allow Server access for User2 in another domain:

Set the Server’s `acl_users` attribute:

```bash
Qmgr: set server acl_users = "-User1@example.com, +*@example.com,
+User2@otherdomain.com"
```

Enable user access control by setting the Server’s `acl_user_enable` attribute to `True`:

```bash
Qmgr: set server acl_user_enable = True
```

Example 8-17: To further require that users of the Server be in Group1 only:

Set the Server’s `acl_groups` attribute:

```bash
Qmgr: set server acl_groups = +Group1
```

Enable group access control by setting the Server’s `acl_group_enable` attribute to `True`:

```bash
Qmgr: set server acl_group_enable = True
```

Example 8-18: To allow access to Queue1 from Host1 only:

Set the queue’s `acl_hosts` attribute:

```bash
Qmgr: set q Queue1 acl_hosts = +Host1@example.com
```

Enable host access control by setting the queue’s `acl_host_enable` attribute to `True`:

```bash
Qmgr: set q Queue1 acl_host_enable = True
```
Chapter 8  Security

8.3.8  Reservation Access

Reservations are designed to be created by unprivileged users, although Managers and Operators can create them as well. The Server’s `resv_enable` attribute controls whether reservations can be created. When this attribute is set to `False`, reservations cannot be created.

Each reservation has its own access control attributes that can be used to specify which users and groups have access to the reservation, and the hosts from which these users and groups are allowed access. The creator of the reservation sets the list of users, groups and hosts that have access to the reservation. This is done at the time of reservation creation, using options to the `pbs_rsub` command.

The creator of a reservation creates that reservation’s ACLs. Reservation ACLs are enabled indirectly through the actions of the reservation creator. The reservation’s list of authorized users is always enabled during reservation creation. The reservation’s lists of authorized groups and authorized hosts are only enabled if explicitly set by the reservation creator.

8.3.8.1  Setting Reservation Access

Reservation access is controlled by the following reservation attributes:

- User access: `Authorized_Users`
- Group access: `Authorized_Groups`
- Host access: `Authorized_Hosts`

The `pbs_rsub` command has three options that allow the user to set the attributes above.

- **-U <authorized user list>**
  
  If the user does not specify this option, `Authorized_Users` is set to the username of the person running the command, and the list is enabled.

  If the user specifies a list of authorized users, `Authorized_Users` is set to this list, and the list is enabled. The creator’s username at the Server is automatically added to this list.

- **-G <authorized group list>**

  If the user does not specify this option, `Authorized_Groups` is left blank, and the list is not enabled.

  If the user specifies a list of authorized groups, `Authorized_Groups` is set to this list, and the list is enabled. The creator’s group at the Server is **not** automatically added to this list.

- **-H <authorized host list>**

  If the user does not specify this option, `Authorized_Hosts` is left blank, and the list is not enabled.
If the user specifies a list of authorized hosts, Authorized_Hosts is set to this list, and the list is enabled. The creator’s host is not automatically added to this list.

Reservation ACLs allow or deny access based on group names, usernames, and host names.

### 8.3.8.2 Reservation Access Defaults

The defaults for each reservation access control attribute are as follows:

- **Authorized_Users**: The reservation creator only is allowed access
- **Authorized_Groups**: All groups are allowed access
- **Authorized_Hosts**: All hosts are allowed access

### 8.3.8.3 Requirements for Reservation Access

If a user is denied access to a reservation via the reservation’s ACLs, that means that the user cannot do the following:

- Submit a job into the reservation
- Move a job into the reservation

However, that user can still operate on his/her own jobs in the reservation, as long as that user is not denied access at the Server level. The user can do the following:

- Delete the job
- Hold the job
- Move the job out of the reservation

For example, if an Operator qmoves User1’s job into a reservation to which User1 is denied access, User1 can still perform operations on the job such as deleting or holding the job, and User1 can qmove the job out of the reservation.

### 8.3.9 How Reservation Access Control Works

Each reservation has its own queue, which has its own ACLs, each controlled by a Boolean switch, as described in section 8.3.7.1, “Server ACLs”, on page 658. Access to a reservation is actually controlled through the reservation’s queue ACLs. Each reservation access control attribute is copied to the equivalent attribute for the reservation’s queue.
8.3.9.1 Creating and Enabling Reservation Queue ACLs

If the group or host reservation ACL is specified by the reservation creator, the associated Boolean switch for the reservation queue ACL is set to True.

Authorized_Users is always set to the creator and copied to the queue’s acl_users attribute, and acl_user_enable is always set to True.

If Authorized_Groups is specified by the creator, it is copied to the queue’s acl_groups attribute and acl_group_enable is set to True. If the reservation creator does not specify a value for Authorized_Groups, nothing is copied to the queue’s acl_groups, and acl_group_enable remains at its default value of False.

If Authorized_Hosts is specified by the creator, it is copied to the queue’s acl_hosts attribute and acl_host_enable is set to True. If the reservation creator does not specify a value for Authorized_Hosts, nothing is copied to the queue’s acl_hosts, and acl_host_enable remains at its default value of False.

The following table shows the relationships between reservation ACLs and reservation queue ACLs:

**Table 8-5: Relationship Between Reservation ACLs and Reservation Queue ACLs**

<table>
<thead>
<tr>
<th>Entity</th>
<th>Reservation ACL</th>
<th>Reservation Queue ACL</th>
<th>Reservation Queue ACL Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>Authorized_Users</td>
<td>acl_users</td>
<td>acl_user_enable</td>
</tr>
<tr>
<td>Groups</td>
<td>Authorized_Groups</td>
<td>acl_groups</td>
<td>acl_group_enable</td>
</tr>
<tr>
<td>Hosts</td>
<td>Authorized_Hosts</td>
<td>acl_hosts</td>
<td>acl_host_enable</td>
</tr>
</tbody>
</table>
8.3.9.2 Examples of Setting Reservation Access

Example 8-19: To disallow access for User1 and allow access for all other users at your domain:

- Set reservation’s Authorized_Users attribute using the -U option to pbs_rsub:
  
  ```bash
  pbs_rsub ... -U "-User1@example.com, +@example.com"
  ```

Example 8-20: To allow access for Group1 and Group2 only:

- Set reservation’s Authorized_Groups attribute using the -G option to pbs_rsub:
  
  ```bash
  pbs_rsub ... -G "+Group1, +Group2"
  ```

Note that any users in Group1 and Group2 to whom you wish to grant access must be explicitly granted access in the Authorized_Users list.

Example 8-21: To allow access from Host1 and Host2 only:

- Set reservation’s Authorized_Hosts attribute using the -H option to pbs_rsub:
  
  ```bash
  pbs_rsub ... -H "+Host1.example.com, +Host2.example.com, -*.example.com"
  ```

8.3.10 Scope of Access Control

Queue-level ACLs provide different security functionality from that provided by Server-level ACLs. Access to PBS commands is controlled by Server-level ACLs. For example, access to the `qstat` and `qselect` operations are controlled only at the Server level. For unprivileged users, access to a specific queue is controlled through that queue’s ACLs.

The users allowed access to a queue or reservation are a subset of the users allowed access to the Server. Therefore, if you wish to allow a user access to a queue, that user must also be allowed access to the Server. The hosts from which a user may run commands at a queue are a subset of the hosts from which a user may run commands at the Server. See “Server Attributes” on page 316 of the PBS Professional Reference Guide, “Queue Attributes” on page 354 of the PBS Professional Reference Guide, and “Reservation Attributes” on page 343 of the PBS Professional Reference Guide.

8.3.11 Operations Controlled by ACLs

ACLs control some operations in PBS, but not others. Manager and Operator privileges override some ACL restrictions.
8.3.1.1 Server Operations Controlled by ACLs

8.3.1.1.i Server Host ACL
If it is enabled, the Server’s host ACL is checked for and controls all Server operations, and is honored regardless of privilege. Any request coming from a disallowed host is denied.

8.3.1.1.ii Server User ACL
If it is enabled, the Server’s user ACL is checked for and controls all Server operations, but is overridden by Manager or Operator privilege. This means that the Server’s user ACL applies only to users, not to Managers or Operators. Even if explicitly denied access in the Server’s user ACL, a PBS Manager or Operator is allowed access to the Server. Note that queue access is controlled separately by queue ACLs; even if Managers or Operators are explicitly denied access in the Server’s user ACL, if a queue’s ACLs are not enabled, Managers and Operators have access to the queue. The same is true for reservations.

8.3.1.2 Queue Operations Controlled by ACLs
If enabled, queue ACLs are applied only when an entity is attempting to enqueue a job. Enqueueing a job can happen in any of three ways:

• Moving a job into the queue
• Submitting a job to the queue
• Routing a job into the queue

Queue ACLs are not applied for non-enqueueing operations, for example:

• Moving a job out of the queue
• Holding a job
• Deleting a job
• Signaling a job
• Getting job status

8.3.1.2.i Queue Host ACL
If a queue’s host ACL is enabled, it is checked when an entity attempts to enqueue a job. The host ACL is always honored, regardless of privilege.

8.3.1.2.ii Queue User and Group ACLs
If a queue’s user or group ACL is enabled, it is applied when an entity attempts to enqueue a job. Manager and Operator privileges override queue user and group ACLs when an entity attempts to move a job into a queue. This means that a PBS Manager or Operator who is
explicitly denied access by the user or group ACL for queue Q1 can still use the qmove command to move a job into Q1, as long as other ACLs allow the operation (the Server’s user and host ACLs must both allow this).

A queue user or group ACL is applied in the following way:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Applied to Users</th>
<th>Applied to Managers/Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moving a job into the queue</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Submitting a job to the queue</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Having a job routed into the queue</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

8.3.11.3 Reservation Operations Controlled by ACLs

Access to a reservation’s queue is controlled through its queue’s ACLs. A reservation’s queue behaves exactly the same way as a regular queue.
### 8.3.11.4 Table of Operations Controlled by ACLs and Overrides

The following table lists which operations are and are not controlled by Server and queue ACLs, and which controls are overridden.

**Table 8-7: Operations Controlled by ACLs, and ACL Overrides**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Server ACLs</th>
<th>Queue ACLs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Host</td>
<td>User</td>
</tr>
<tr>
<td>Move job into queue</td>
<td>Y Y N N Y Y Y</td>
<td>Y N N Y Y Y Y Y</td>
</tr>
<tr>
<td>Move job out of queue</td>
<td>Y Y N N Y Y Y</td>
<td>N - - N - - N - -</td>
</tr>
<tr>
<td>Submit job to queue</td>
<td>Y Y N N Y Y Y</td>
<td>Y N N Y N Y N N</td>
</tr>
<tr>
<td>Have job routed into queue</td>
<td>Y Y N N Y Y Y</td>
<td>Y N N Y N Y N N</td>
</tr>
<tr>
<td>Delete job</td>
<td>Y Y N N Y Y Y</td>
<td>N - - N - - N - -</td>
</tr>
<tr>
<td>Hold job</td>
<td>Y Y N N Y Y Y</td>
<td>N - - N - - N - -</td>
</tr>
<tr>
<td>Release job</td>
<td>Y Y N N Y Y Y</td>
<td>N - - N - - N - -</td>
</tr>
<tr>
<td>Signal job</td>
<td>Y Y N N Y Y Y</td>
<td>N - - N - - N - -</td>
</tr>
<tr>
<td>Status job</td>
<td>Y Y N N Y Y Y</td>
<td>N - - N - - N - -</td>
</tr>
<tr>
<td>Status Server</td>
<td>Y Y N N Y Y Y</td>
<td>N - - N - - N - -</td>
</tr>
<tr>
<td>Status queue</td>
<td>Y Y N N Y Y Y</td>
<td>N - - N - - N - -</td>
</tr>
</tbody>
</table>

### 8.3.12 Avoiding Problems

#### 8.3.12.1 Using Group Lists

When a user specifies a group list, each and every group in which that user might execute a job must have a group name and an entry in the groups database, for example, `/etc/group.`
8.3.13 Flatuid and Access

The Server’s flatuid attribute affects both when users can operate on jobs and whether users without accounts on the Server host can submit jobs.

8.3.13.1 How flatuid Controls When Users Can Operate On Jobs

This section describes how the Server’s flatuid attribute affects the circumstances under which users can operate on jobs.

This attribute specifies whether, for each user, the username at the submission host must be the same as the one at the Server host. The username at the Server host must always be the same as the username at the execution host. When flatuid is set to True, the Server assumes that UserA@host1 is the same as UserA@host2. Therefore, if flatuid is True, UserA@host2 can operate on UserA@host1’s job.

The value of flatuid also affects whether .rhosts and host.equiv are checked. If flatuid is True, .rhosts and host.equiv are not queried, and for any users at host2, only UserA is treated as UserA@host1. If flatuid is False, .rhosts and host.equiv are queried.

That is, when flatuid is True, even if UserB@host2 is in UserA@host1’s .rhosts, UserB@host2 cannot operate on UserA’s job(s). If flatuid is False, and UserB@host2 is in UserA@host1’s .rhosts, UserB@host2 is allowed to operate on UserA’s job(s).

Example:

- UserA@host1 has a job
- UserB@host2 is in UserA@host1’s .rhosts
  a. flatuid = True: UserB@host2 cannot operate on UserA’s job
  b. flatuid = False: UserB@host2 can operate on UserA’s job

The following table shows how access is affected by both the value of the server’s flatuid attribute and whether UserB@host2 is in UserA@host1’s .rhosts:

### Table 8-8: Effect of flatuid Value on Access

<table>
<thead>
<tr>
<th>flatuid</th>
<th>UserB@host2 in UserA@host1’s .rhosts</th>
<th>Is UserA@host1 treated as UserA@host2?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

| flatuid = True | Yes | No |
| flatuid = False | Yes | No |
8.3.13.2 How flatuid Affects Users Without Server Accounts

This section describes how the Server’s flatuid attribute affects users who have no account on the Server host.

- If flatuid is set to False, users who have no account at the Server host cannot submit jobs to PBS.
- If flatuid is set to True, these users can submit jobs. However, the job will only run if it is sent to execution hosts where the user does have an account. If the job is sent to execution hosts where the user does not have an account, the job will not run, and the MoM will log an error message.

8.4 Restricting Execution Host Access

You can configure each PBS execution host so that the only users who have access to the machine are those who are running jobs on the machine. You can specify this by adding the restrict_user parameter to the MoM configuration file PBS_HOME/mom_priv/config. This parameter is a Boolean, which if set to True, prevents any user not running a job from running any process on the machine for more than 10 seconds. The interval between when PBS applies restrictions depends upon MoM’s other activities, but can be no more than 10 seconds.

You can specify which users are exempt from this restriction by adding the restrict_user_exceptions parameter to the same file. See the description of the parameter in the next section.

You can allow system processes to run by specifying the maximum numeric user ID allowed access to the machine when not running a job. You do this by adding the restrict_user_maxsysid parameter to the MoM configuration file. PBS automatically tries to allow system processes to run: if restrict_user is enabled and restrict_user_maxsysid is unset, PBS looks in /etc/login.defs for SYSTEM_UID_MAX for the value to
use. If there is no maximum ID set there, it looks for SYSTEM_MIN_UID, and uses that value minus 1. Otherwise PBS uses the default value of 999. See section 3.6.6, “Restricting User Access to Execution Hosts”, on page 59 and “$restrict_user <value>” on page 276 of the PBS Professional Reference Guide.

8.4.1 MoM Access Configuration Parameters

These are the configuration parameters in PBS_HOME/mom_priv/config that can be set to restrict and specify access to each execution host. Each execution host has its own configuration file.

$restrict_user <value>
Controls whether users not submitting jobs have access to this machine. When True, only those users running jobs are allowed access.
Format: Boolean
Default: off

$restrict_user_exceptions <user_list>
List of users who are exempt from access restrictions applied by $restrict_user.
Maximum number of names in list is 10.
Format: Comma-separated list of usernames; space allowed after comma

$restrict_user_maxsysid <value>
Allows system processes to run when $restrict_user is enabled. Any user with a numeric user ID less than or equal to value is exempt from restrictions applied by $restrict_user.
Format: Integer
Default: 999

8.4.2 Examples of Restricting Access

To restrict user access to those running jobs, add:

$restrict_user True

To specify the users who are allowed access whether or not they are running jobs, add:

$restrict_user_exceptions <user list>

For example:

$restrict_user_exceptions User1, User2
Chapter 8

To allow system processes to run, specify the maximum numeric user ID by adding:

```
$restrict_user_maxsysid <user ID>
```

For example:

```
$restrict_user_maxsysid 999
```

### 8.5 Logging Security Events

Each PBS daemon logs security-related events, at event class 32 (0x0020) or at event class 128 (0x0080). For information about daemon logfiles, see section 12.4, “Event Logging”, on page 867.

#### 8.5.1 Events Logged at Event Class 32 (0x0020)

The following security-related events are logged at decimal event class 32 (0x0020):

- When an execution host has access restrictions in place via the `$restrict_user` configuration parameter, and MoM detects that a user who is not exempt from access restriction is running a process on the execution host, MoM kills that user's processes and writes a log message:

  ```
  01/16/2006 22:50:16;0002;pbs_mom;Svr;restrict_user;
  killed uid 1001 pid 13397(bash) with log event class PBSE_SYSTEM.
  ```


- If for some reason the access permissions on the PBS file tree are changed from their default settings, a daemon may detect this as a security violation, refuse to execute, and write an error message in the corresponding log file. The following are examples of each daemon's log entry:

  ```
  Server@<host>: Permission denied (13) in chk_file_sec, Security violation
  "/usr/spool/PBS/server_priv/jobs/" resolves to "/usr/spool/PBS"
  pbs_mom: Permission denied (13) in chk_file_sec, Security violation "/usr/
  spool/PBS/mom_priv/jobs/" resolves to "/usr/spool/PBS"
  pbs_sched: Permission denied (13) in chk_file_sec, Security violation "/
  _usr/spool/PBS/sched_priv" resolves to "/usr/spool/PBS"
  ```

  A Manager can run `pbs_probe` (on UNIX/Linux) or `pbs_mkdirs` (on Windows) to check and optionally correct any directory permission or ownership problems.

- When a user without a password entry (an account) on the Server attempts to submit a
job, the Server logs this event. The following is an example log entry:
8/21/2009 15:28:30;0080;Server@capella;Req;req_reject;Reject reply
code=15023, aux=0, type=1, from User1@host1.example.com

- If a daemon detects that a file or directory in the PBS hierarchy is a symbolic link pointing to a non-secure location, this is written to the daemon’s log. The resulting log message is the same as for a permission violation:
  Server@<host>: Permission denied (13) in chk_file_sec, Security violation
  
  "/usr/spool/PBS/server_priv/jobs/" resolves to "/usr/spool/PBS"
  pbs_mom: Permission denied (13) in chk_file_sec, Security violation 
  "/usr/spool/PBS/mom_priv/jobs/" resolves to "/usr/spool/PBS"
  pbs_sched: Permission denied (13) in chk_file_sec, Security violation 
  "/usr/spool/PBS/sched_priv" resolves to "/usr/spool/PBS"

- If an $action script is to be executed for a job belonging to a user who does not have an account on an execution host, the execution host’s MoM logs this event. The following is an example log entry:
  08/21/2009 16:06:49;0028;pbs_mom;Job;2.host1;No Password Entry for User User1

- When a job triggers an action script for which the environment cannot be set up, perhaps due to a system error, the MoM attempting to run the action script logs the event. The log message contains the following:
  jobid:failed to setup dependent environment!

- When the scheduler attempts to run a job on an execution host where the job’s owner does not have an account, the MoM on the execution host logs this event. The following is an example log entry:
  08/21/2009 15:51:14;0028;pbs_mom;Job;1.host1;No Password Entry for User User1

- When the scheduler attempts to run a job on an execution host where the job’s owner does not have a home directory, and when the job’s sandbox attribute is not set to PRIVATE, the execution host’s MoM logs this event. The log message contains the following:
  Access from host not allowed, or unknown host: <numeric IP address>
  See “pbs_mom” on page 58 of the PBS Professional Reference Guide.

- If an attempt is made to connect to a host in the PBS complex from an unknown host, the PBS daemon logs the information at both levels 32 and 128 (0x0020 and 0080).
8.5.1.1 Events Logged at Event Class 128 (0x0080)

The following security-related event is logged at event class 128 (0x0080):

- If an attempt is made to connect to a host in the PBS complex from an unknown host, the PBS daemon logs the information at both levels 32 and 128 (0x0020 and 0080).
- If a user or Operator tries to set an attribute that can be set by Managers only, or attempts to create or delete vnodes:
  
  The qmgr command returns this error message:

  qmgr obj=<object> svr=default: Unauthorized Request
  qmgr: Error (15007) returned from server

  The Server logs the following message:

  Req;req_reject;Reject reply code=15007, aux=0, type=9, from <username>

- When a user is denied access to the Server because of the contents of the acl_users Server attribute, the Server logs the following:

  Req;req_reject;Reject reply code=15007, aux=0, type=21, from username@host.domain.com

8.5.1.2 Events Logged at Event Class 1

- When an attempt is made to contact MoM from a non-privileged port for a request requiring a privileged port, MoM logs the following:

  pbs_mom;Svr;pbs_mom;Unknown error: 0 (0) in rm_request, bad attempt to connect message refused from port 61558 addr 127.0.0.1

8.5.1.3 Events Not Logged

The following events are not logged:

- When an attempt is made to connect to a host in the PBS complex from a disallowed host
- When an ACL check denies an entity access to a PBS object
- A user tries to query other users’ jobs when the Server’s query_other_jobs attribute is set to False
- When an Operator or Manager overrides the Server’s user ACL
8.6 Changing the PBS Service Account Password

Normally, the password for the PBS service account on Windows should not be changed. But if it is necessary to change it, perhaps due to a security breach, then do so using the following steps:

1. Change the PBS service account’s password on one machine in a command prompt from an admin-type of account by typing:

   Domain environments:
   
   ```
   net user <name of PBS service account> * /domain
   ```
   
   Non-domain environment:
   
   ```
   net user <name of PBS service account> *
   ```

2. Provide the Service Control Manager (SCM) with the new password given above. Do this either using the GUI-based Services application which is one of the Administrative Tools, or by unregistering and re-registering the PBS services with the password. See “pbs_account” on page 42 of the PBS Professional Reference Guide.

   To unregister on 32-bit Windows systems:
   
   ```
   pbs_account --unreg "\Program Files\PBS Pro\exec\sbin\pbs_server.exe"
   pbs_account --unreg "\Program Files\PBS Pro\exec\sbin\pbs_mom.exe"
   pbs_account --unreg "\Program Files\PBS Pro\exec\sbin\pbs_sched.exe"
   pbs_account --unreg "\Program Files\PBS Pro\exec\sbin\pbs_rshd.exe"
   ```

   To re-register on 32-bit Windows systems:
   
   ```
   pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_server.exe"
   pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_mom.exe"
   pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_sched.exe"
   pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_rshd.exe"
   ```
To unregister on 64-bit Windows systems:

```
pbs_account --unreg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_server.exe"
pbs_account --unreg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_mom.exe"
pbs_account --unreg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_sched.exe"
pbs_account --unreg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_rshd.exe"
```

To re-register on 64-bit Windows systems:

```
pbs_account --reg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_server.exe"
pbs_account --reg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_mom.exe"
pbs_account --reg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_sched.exe"
pbs_account --reg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_rshd.exe"
```

When re-registering, you can give an additional `-p password` argument to the `pbs_account` command, to specify the password on the command line.

### 8.7 Paths and Environment Variables

A significant effort has been made to ensure the various PBS components themselves cannot be a target of opportunity in an attack on the system. The two major parts of this effort are the security of files used by PBS and the security of the environment. Any file used by PBS, especially files that specify configuration or other programs to be run, must be secure. The files must be owned by root and in general cannot be writable by anyone other than root.
A corrupted environment is another source of attack on a system. To prevent this type of attack, each component resets its environment when it starts. If it does not already exist, the environment file is created during the install process. As built by the install process, it will contain a very basic path and, if found in root's environment, the following variables:

- TZ
- LANG
- LC_ALL
- LC_COLLATE
- LC_CTYPE
- LC_MONETARY
- LC_NUMERIC
- LC_TIME

The environment file may be edited to include the other variables required on your system.

The entries in the PBS_ENVIRONMENT file can take two possible forms:

```
variable_name=value
variable_name
```

In the latter case, the value for the variable is obtained before the environment is reset.

### 8.7.1 Path Caveats

Note that PATH must be included. This value of PATH will be passed on to batch jobs. To maintain security, it is important that PATH be restricted to known, safe directories. Do NOT include "." in PATH. Another variable which can be dangerous and should not be set is IFS.

### 8.8 File and Directory Permissions

Each parent directory above PBS_HOME must be owned by root and writable by root only. All files and directories used by PBS should be writable by root only. Permissions should allow read access for all files and directories except those that are private to the daemons. The following should not be writable by any but root:

```
PBS_HOME/mom_priv
PBS_HOME/sched_priv
PBS_HOME/server_priv
```
The `PBS_HOME` directory must be readable and writable from server hosts by root (Administrator) on UNIX/Linux.

On Windows, `PBS_HOME` must have Full Control permissions for the local "Administrators" group on the local host.

PBS checks permissions for certain files and directories. The following error message is printed for certain files and directories (e.g. `/etc/pbs.conf`, `/var/spool/PBS/mom_priv/config`, etc.) if their permissions present a security risk:

```
<command>: Not owner (1) in chk_file_sec, Security violation "<directory>"
        resolves to "<directory>"
<command>: Unable to configure temporary directory.
```

### 8.9 Authentication & Authorization

#### 8.9.1 Host Authentication

PBS uses a combination of information to authenticate a host. If a request is made from a client whose socket is bound to a privileged port (less than 1024, which requires root privilege), PBS believes the IP (Internet Protocol) network layer as to whom the host is. If the client request is from a non-privileged port, the name of the host which is making a client request must be included in the credential sent with the request and it must match the IP network layer opinion as to the host’s identity.

#### 8.9.2 Host Authorization

Access to the Server from another system may be controlled by an access control list (ACL). Access to `pbs_mom` is controlled through a list of hosts specified in the `pbs_mom`’s configuration file. By default, only “localhost”, the name returned by `gethostname(2)`, and the host named by `PBS_SERVER` from `/etc/pbs.conf` are allowed. See “MoM Parameters” on page 267 of the PBS Professional Reference Guide for more information on the configuration file. Access to `pbs_sched` is not limited other than it must be from a privileged port.

#### 8.9.3 User Authentication

The PBS Server authenticates the username included in a request using the supplied PBS credential. This credential is supplied by `pbs_iff`.
8.9.4 Enabling Host-based Authentication on Linux

Host-based authentication will allow users within your complex to execute commands on or transfer files to remote machines. This can be accomplished for both the r-commands (e.g., rsh, rcp), and secure-commands (e.g., ssh, scp). The following procedure does not enable root to execute any r-commands or secure-commands without a password. Further configuration of the root account would be required.

Correct name resolution is important. Using fully qualified domain names on one machine and short names on another will not work. Name resolution must be consistent across all machines.

8.9.4.1 RSH/RCP

- Verify that the rsh-server and rsh-client packages are installed on each host within the complex.
- Verify that the rsh and rlogin services are on on each host within the complex. Example:
  ```bash
  chkconfig --list | grep -e rsh -e rlogin
  rlogin: on
  rsh:    on
  ```

On the headnode (for simplicity) add the hostname of each host within the complex to `/etc/hosts.equiv`, and distribute it to each host within the complex. Example file (filename: `/etc/hosts.equiv`):

```
headnode
node01
node02
node03
node04
node05
```

8.9.4.2 SSH/SCP

- Verify that the openSSH package is installed on each host within the complex.
- Verify that the openSSH service is on on each host within the complex. Example:
  ```bash
  chkconfig --list | grep ssh
  sshd 0:off 1:off 2:on 3:on 4:on 5:on 6:off
  ```
- Modify the following `ssh_config` files on each host within the complex to enable the host-based authentication. These options may be commented out, and so must be uncom-
Chapter 8  

Security

mented and set.

a. /etc/ssh/sshd_config  
   HostbasedAuthentication yes

b. /etc/ssh/ssh_config  
   HostbasedAuthentication yes

- Stop and start the openSSH service on each host within the complex.
   
   /etc/init.d/sshd stop
   
   /etc/init.d/sshd start

- On the headnode (for simplicity) create a file which contains the hostname and IP address of each host within the complex, where the hostname and IP address are comma delimited. Each entry should have all of the information from the line in /etc/hosts.

Example file (filename: ssh_hosts):

headnode,headnode.company.com,192.168.1.100  
node01,node01.company.com,192.168.1.1  
node02,node02.company.com,192.168.1.2  
node03,node03.company.com,192.168.1.3  
node04,node04.company.com,192.168.1.4  
node05,node05.company.com,192.168.1.5

So that if your /etc/hosts file has:

192.168.1.7 host05.company.com host05  

the line in ssh_hosts would be:

node05,node05.company.com,192.168.1.7

- Gather each host’s public ssh host key within the complex by executing ssh-keyscan against the ssh_hosts file created in Step 5, and distribute the output to each host within the complex.

   ssh-keyscan -t rsa -f ssh_hosts > /etc/ssh/ssh_known_hosts2

- Create the /etc/ssh/hosts.equiv file for all of the machines in the complex. This must list the first name given in each line in the /etc/hosts file. Using the example from step 5:

Your /etc/hosts file has:

192.168.1.7 host05.company.com host05

The hosts.equiv file should have:

node05.company.com

- Every machine in the complex will need to have ssh_config and sshd_config
updated. These files can be copied out to each machine.

### 8.9.4.3 Special Notes

The configurations of OpenSSH change (frequently). Therefore, it is important to understand what you need to set up. Here are some tips on some versions.

#### 8.9.4.3.i OpenSSH_3.5p1

Procedure above should work.

#### 8.9.4.3.ii OpenSSH_3.6.1p2

Procedure above should work with the following additional step:

Define “EnableSSHKeysign yes” in the `/etc/ssh/ssh_config` file

#### 8.9.4.3.iii OpenSSH_3.9p1

Procedure above should work with the following two additional steps:

1. Define “EnableSSHKeysign yes” in the `/etc/ssh/ssh_config` file
2. Change permissions from 0755 to 4755:

```bash
chmod 4755 /usr/lib/ssh/ssh-keysign
```

This file is required to be setuid to work.

#### 8.9.4.3.iv LAM

Use “`ssh -x`” instead of “`ssh`”.

If you want to use SSH you should enable ‘PermitUserEnvironment yes’ so that the user's environment will be passed to the other hosts within the complex. Otherwise, you will see an issue with `tkill` not being in the user's PATH when executing across the hosts.

#### 8.9.4.3.v SuSe

Be sure that hostname resolution works correctly. Double-check that the server machine resolves the client’s IP address to the same name that is in your known hosts and `shosts.equiv`. You can have comma-delimited names, including IP addresses, ahead of the key, as in the following:

```
example, example.domain, 192.168.100.1 ssh-rsa thekey
```

Performing host based authentication as root may require that you set up a `.shosts` file in the home directory of root.

See [http://www.ssh.com/support/documentation/online/ssh/admin-guide/32/Host-Based_Authentication.html](http://www.ssh.com/support/documentation/online/ssh/admin-guide/32/Host-Based_Authentication.html) for more information.
8.10 Root-owned Jobs

The Server will reject any job which would execute under the UID of zero unless the owner of the job, typically root/Administrator, is listed in the Server attribute acl_roots.

The Windows version of PBS considers as a “root” account the following:

• Local SYSTEM account
• Account that is a member of the local Administrators group on the local host
• Account that is a member of the Domain Admins group on the domain
• Account that is a member of the Administrators group on the domain controller
• Account that is a member of the Enterprise Admins group on the domain
• Account that is a member of the Schema Admins group on the domain

In order to submit a job from this “root” account on the local host, be sure to set acl_roots.

For instance, if user foo is a member of the Administrators group, then you need to set:

Qmgr: set server acl_roots += foo

in order to submit jobs and not get a “bad UID for job execution” message.

8.10.1 Caveats for Root-owned Jobs

Allowing “root” jobs means that they can run on a configured host under the same account which could also be a privileged account on that host.

8.11 User Passwords

PBS has different password requirements dictated by the UNIX, Linux and Windows operating systems. Jobs submitted on UNIX and Linux systems do not require passwords. Jobs on Windows systems require passwords.

See the PBS Professional 12.2 release notes for a list of supported architectures.

PBS provides two systems for handling user passwords, but these two systems are incompatible. These two systems are per-user/per-server passwords and per-job passwords.
8.11.1 Windows

Windows systems require a password for PBS to run a process as the user, so users on these systems must supply a password. User passwords can be managed using one of the following:

- Per-user/per-server passwords, where each user submits a password once for each server. Note that this method cannot be used when the PBS complex contains machines that are not Windows submission hosts.
- Per-job passwords, where each user submits a password once for each job.

You can choose one or the other, but not both.

8.11.1.1 Per-user/per-server Passwords

You can configure the PBS server to store user passwords, so that each user only has to submit a password once, and that password is used for all of the user’s jobs. To enable the server to store user passwords, set the server’s single_signon_password_enable attribute to True. The pbs_password command is then run for each user, either by the user or by you, before the user submits any jobs.

8.11.1.1.i The single_signon_password_enable Server Attribute

    single_signon_password_enable
    If enabled, this option allows users to specify their passwords only once, and PBS will remember them for future job executions. Incompatible with qsub -Wpwd.
    Format: Boolean
    Default: False (UNIX), True (Windows)
    Python attribute value type: bool

8.11.1.1.ii Examples

To enable the server to store passwords:

    Qmgr: set server single_signon_password_enable = True

To specify a user’s password:

    pbs_password

8.11.1.1.iii Caveats for single_signon_password_enable

- When the server’s single_signon_password_enable attribute is set to True, users cannot submit jobs from any non-Windows, non-Linux machine. This is because the
Chapter 8  

Security

The `pbs_password` command is not available on those machines.

- When the server’s `single_signon_password_enable` attribute is set to `True`, users cannot use the `-Wpwd` option to the `qsub` command.
- The server’s `single_signon_password_enable` attribute can only be set when there are no jobs, or all jobs have a “bad password” hold, and can only be disabled when no jobs exist.
- A user must set a password at any server at which the user wishes to run jobs, if that server has `single_signon_password_enable` set to `True`. For example, if a user wishes to move a job to another server via the `qmove` command, and the receiving server has `single_signon_password_enable` set to `True`, the user’s password must first be stored at the receiving server via the `pbs_password` command. This applies to jobs moved for peer scheduling as well.

8.11.1.1.iv Single Signon and Invalid Passwords

If a job’s originating Server has `single_signon_password_enable` set to `True`, and the job fails to run due to a bad password, the Server will place a hold on the job of type “p” (bad password hold), update the job’s comment with the reason for the hold, and email the user with possible remedy actions. The user (or a manager) can release this hold type via:

```
qrls -h p <jobid>
```

8.11.1.2 Per-job Passwords

If your PBS complex contains non-Windows, non-Linux machines, use per-job passwords. Users can submit a password for each job that requires a password by using the `-Wpwd` option to the `qsub` command. For example:

```
qsub -lselect = 2:mem=2GB -Wpwd
```

Users cannot use this method when the server’s `single_signon_password_enable` attribute is set to `True`.

8.11.2 UNIX and Linux Only

If your PBS complex contains only UNIX and Linux machines, you do not need user passwords. Make sure that the server’s `single_signon_password_enable` attribute is set to `False` or is unset, otherwise some jobs will not run.
8.11.3 Changing the PBS Service Account Password

Normally, the PBS service account password must not be changed. But if it is deemed necessary to change it perhaps due to a security breach, then do so using the following steps:

First, change the PBS service account's password on a machine in a command prompt from an admin-type of account by typing:

Domain environments:

```
net user <name of PBS service account> * /domain
```

Non-domain environment:

```
net user <name of PBS service account> *
```

Then the Service Control Manager (SCM) must be provided with the new password specified above. This can be done via the GUI-based Services application found as one of the Administrative Tools, or unregister and re-register the PBS services with password.

On 32-bit Windows systems:

```
pbs_account --unreg "\Program Files\PBS Pro\exec\sbin\pbs_server.exe"
pbs_account --unreg "\Program Files\PBS Pro\exec\sbin\pbs_mom.exe"
pbs_account --unreg "\Program Files\PBS Pro\exec\sbin\pbs_sched.exe"
pbs_account --unreg "\Program Files\PBS Pro\exec\sbin\pbs_rshd.exe"
```

```
pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_server.exe"
pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_mom.exe"
pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_sched.exe"
pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_rshd.exe"
```

On 64-bit Windows systems:

```
pbs_account --unreg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_server.exe"
pbs_account --unreg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_mom.exe"
pbs_account --unreg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_sched.exe"
pbs_account --unreg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_rshd.exe"
```

```
pbs_account --reg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_server.exe"
pbs_account --reg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_mom.exe"
pbs_account --reg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_sched.exe"
pbs_account --reg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_rshd.exe"
```
The register form (last four lines above) can take an additional argument 
-p password so that you can specify the password on the command line directly.

8.12 Windows Caveats

The PBS installer installs the Microsoft 2005 redistributable pack. Please refer to the Microsoft documentation for further details on this package.

8.13 Windows XP SP2 Firewall

Under Windows XP Service Pack 2 (SP2) the Windows Firewall may have been turned on by default. If so, it will block incoming network connections to all services including PBS. Therefore after installing PBS Professional, to allow pbs_server, pbs_mom, pbs_sched, and pbs_rshd to accept incoming connections:

Access Settings->Control Panel->Security Center->Windows Firewall, and verify that the Windows Firewall has been set to “ON” to block incoming network connections.

From this panel, you can either turn Windows Firewall “off”, or click on the Exceptions tab and add the following to the list:

[INSTALL PATH]\exec\sbin\pbs_server.exe
[INSTALL PATH]\exec\sbin\pbs_mom.exe
[INSTALL PATH]\exec\sbin\pbs_sched.exe
[INSTALL PATH]\exec\sbin\pbs_rshd.exe

8.14 Windows Requirement for cmd Prompt

On Windows 7 and later with UAC enabled, if you will use the cmd prompt to operate on hooks, or for any privileged command such as qmgr, you must run the cmd prompt with option Run as Administrator.
Chapter 9

Making Your Site More Robust

This chapter describes how to configure PBS to make your site more robust. If PBS is not already installed, install it according to the PBS Professional Installation & Upgrade Guide.

9.1 Robustness

PBS provides the following mechanisms that support site robustness and flexibility:

   Failover
       The PBS complex can run a backup server. If the primary server fails, the secondary takes over without an interruption in service.

   Checkpoint/Restart
       Allows jobs to be checkpointed and restarted. Uses OS-provided or third-party checkpoint/restart facility.

   Controls for Communication, Timing, and Load
       PBS allows setting parameters to prevent problems in communication, timing, and load on vnodes.

   Reservation Fault Tolerance
       PBS attempts to ensure that reservations run by finding usable vnodes when reservation vnodes become unavailable.

   OOM Killer Protection
       PBS is installed so that daemons are protected from an OOM killer.
9.2 Failover

9.2.1 Terminology

**Primary Server**
The PBS Professional server daemon which is running during normal operation.

**Secondary Server**
The PBS Professional server daemon which takes over when the primary server fails.

**Primary Scheduler**
The PBS Professional scheduler daemon which is running during normal operation.

**Secondary Scheduler**
The PBS Professional scheduler daemon which takes over when the primary scheduler is not available.

**Active**
A server daemon is active when it is managing user requests and communicating with the scheduler and MoMs.

**Idle**
A server daemon is idle when it is running, but only accepting handshake messages, not performing workload management.

9.2.2 How Failover Works

During normal operation, the primary server is active and the secondary server is idle. If the primary server fails for any reason, the secondary server becomes active and takes over server functions for the complex. No work is lost during the transition between servers. PBS functions the same during failover as it does during normal operation. The PBS data service is considered to be part of the PBS server; if it fails, this triggers failover.

9.2.2.1 Primary and Secondary Schedulers

Each server is paired with its own scheduler. The primary server can use only the primary scheduler, but the secondary server can use either scheduler. The secondary server will use the primary scheduler if possible, but will start its own scheduler if necessary.
9.2.2.1.i  Starting Secondary Scheduler Under Windows

The following information is also included in the configuration instructions later; it is here for explanation only. Under Windows, the secondary server only starts a scheduler if it needs to when it takes over from the primary server, and not when the host is booted. To achieve this, the `pbs.conf` file on the secondary server host contains the line, “PBS_START_SCHED=1”, which is different from UNIX/Linux, and the startup type for the scheduler is “Manual”. The server starts the scheduler manually using “net start pbs_sched”.

9.2.2.2  Primary and Secondary Data Services

Each server uses a data service. The primary server uses only its own data service. The secondary server first attempts to use the primary data service, but will start its own if necessary.

9.2.2.3  Normal Post-configuration Behavior

After you have configured PBS for failover, and started both servers, the secondary server periodically attempts to connect to the primary server until it succeeds and registers itself with the primary server. The secondary server must register itself in order to take over upon failure of the primary server.

9.2.2.4  Behavior During Failover

When both server daemons are running, the primary server sends periodic handshake messages to the secondary. If the secondary server stops receiving handshake messages from the primary server, the following happens:

- The secondary server waits for a specified delay period before taking over. This delay is specified using the `pbs_server -F` option. The default period is 30 seconds.
  - The secondary server records the timestamp of the `PBS_HOME/server_priv/svrlive` file
  - The secondary waits for the specified delay, then checks the time stamp again
  - If the time stamp has changed, the secondary server remains idle
  - If the time stamp has not changed, the secondary attempts to open a new TCP connection to the primary
  - If the secondary server cannot open a TCP connection to the primary, the secondary becomes active

- The secondary server logs a message saying that failover has occurred.
- An email is sent to and from the account defined in the server’s `mail_from` attribute, say-
The secondary server checks whether the primary scheduler is still running.
- If the primary scheduler is still running, the secondary server uses it.
- If the primary scheduler is not running, the secondary server starts the secondary scheduler.

The secondary server attempts to connect to the data service:
- The secondary server tries for 20 seconds to connect to the data service on the primary server’s host.
- If it cannot, it tries for 20 seconds to connect to a data service on the secondary server’s host.
- If the secondary server cannot connect to a data service on either host, it attempts to start a data service on the secondary server’s host.

The secondary server notifies all of the MoMs that it is the active server.

The secondary server begins responding to network connections and accepting requests from client commands such as `qstat` and `qsub`.

### 9.2.2.5 Behavior When Primary Resumes Control

When the primary server starts back up, it takes control from the secondary server, becoming the active server. The secondary server becomes idle and resumes listening for the regular handshake messages from the primary server.

The primary server may have been stopped for any of several reasons. The restart method will vary accordingly. If the host was stopped, the PBS server is restarted automatically when the host is started. If the host is still up but the server was stopped, restart the server. See “Manually Starting the Server” on page 181 in the PBS Professional Installation & Upgrade Guide.

The primary server uses only its own scheduler and data service. When the primary server resumes control, it starts a data service, and if the secondary scheduler and/or data service is running, they are stopped. No data is lost in the transition.

When the primary has taken control, the secondary logs a message saying so:

```
received takeover message from primary, going inactive
```

### 9.2.2.6 Server Name and Job IDs During Failover

The server name and job IDs do not change when the secondary server is active. For example, the primary server is on a host named `PrimaryHost.example.com`, and the secondary server is on a host named `SecondaryHost.example.com`. When the primary server is
active, the server name is *PrimaryHost*, jobs are given job IDs of the form *NNNN.PrimaryHost*, and the value of the *server_host* server attribute is *PrimaryHost.example.com*. When the secondary server is active, the server name is still *PrimaryHost*, jobs are still given job IDs of the form *NNNN.PrimaryHost*, but the value of *server_host* is *SecondaryHost.example.com*.

The table below summarizes the server name, value of *server_host* and the IDs given to jobs, when either the primary or secondary server is active.

<table>
<thead>
<tr>
<th>Active Server</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hostname</strong></td>
<td></td>
</tr>
<tr>
<td>Primary</td>
<td><em>PrimaryHost.example.com</em></td>
</tr>
<tr>
<td>Secondary</td>
<td><em>SecondaryHost.example.com</em></td>
</tr>
<tr>
<td><strong>Server Name</strong></td>
<td></td>
</tr>
<tr>
<td>Primary</td>
<td><em>PrimaryHost</em></td>
</tr>
<tr>
<td>Secondary</td>
<td><em>PrimaryHost</em></td>
</tr>
<tr>
<td><strong>Value of server_host</strong></td>
<td></td>
</tr>
<tr>
<td>Primary</td>
<td><em>PrimaryHost.example.com</em></td>
</tr>
<tr>
<td>Secondary</td>
<td><em>SecondaryHost.example.com</em></td>
</tr>
<tr>
<td><strong>Job Name</strong></td>
<td></td>
</tr>
<tr>
<td>Primary</td>
<td><em>NNNN.PrimaryHost</em></td>
</tr>
<tr>
<td>Secondary</td>
<td><em>NNNN.PrimaryHost</em></td>
</tr>
</tbody>
</table>

### 9.2.2.7 Information Used by Primary and Secondary Servers

The primary and secondary servers share a single source for attribute information, so anything set via the `qmgr` command need only be set once. *PBS_HOME* is in a shared location. License information is shared and needs to be set at only one server.

Each server, execution and client host uses its own *pbs.conf* file, so these must be set for each host in the complex.

### 9.2.2.8 Impact on Users

Users will not notice when a failover occurs. When a user uses a PBS command such as `qstat`, the command tries to connect to the primary server first. If that fails, the command tries the secondary server.

If the secondary server responds to the command, the command creates a local file so that this process is not repeated for every PBS command.
Chapter 9  Making Your Site More Robust

Under UNIX, the file is named:

/tmp/.pbsrc.UID

where UID is the user ID.

Under Windows, the file is named:

%TEMP\pbsrc.USERNAME

where USERNAME is the user’s login name.

When this file exists, commands try the secondary server first, eliminating the delay in attempting to connect to the down server. If a command cannot connect to the secondary server, and can connect to the primary server, the command removes the file.

The file is removed when the primary server takes over.

9.2.2.9  Determining Which Server Is Active

The server attribute server_host contains the name of the host on which the active server is running. Use the qstat -Bf command to see the value of server_host.

9.2.2.10  Delay Between Primary Failure and Secondary Becoming Active

The default delay time from detection of possible primary server failure until the secondary server takes over is 30 seconds. A secondary server on a very reliable network can use a shorter delay. A secondary server on an unreliable network may need to use a longer delay. The delay is specified via the -F option to the pbs_server command.

9.2.2.11  Communication with MoMs

- If a MoM will see different server addresses, add a $clienthost entry to MoM’s configuration file for each possible server address.
- The secondary server is automatically added to the list of hosts allowed to connect to MoMs, in the $clienthost MoM configuration parameter.

9.2.3  Windows Locations

Since PBS is installed on 32-bit Windows systems in \Program Files\PBS Pro\, and on 64-bit Windows systems in \Program Files (x86)\PBS Pro\, we use INSTALLATION_BASE to indicate either \Program Files\PBS Pro\ or \Program Files (x86)\PBS Pro\, whichever is appropriate.
### 9.2.4 Prerequisites for Failover

#### 9.2.4.1 Checklist of Prerequisites for Failover

The following table contains a checklist of the prerequisites for failover. Each entry has a link to more detailed information about the entry.

**Table 9-2: Prerequisites for Failover**

<table>
<thead>
<tr>
<th>Prerequisite</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identical server hosts</td>
<td>See section 9.2.4.2, “Server Host Requirements”, on page 692</td>
</tr>
<tr>
<td>MoMs on server hosts don’t share a mom_priv directory</td>
<td>See section 9.2.4.3, “Requirements for MoMs on Server Hosts”, on page 692</td>
</tr>
<tr>
<td>All hosts must be able to communicate over the network</td>
<td>See section 9.2.4.4, “Communication Between Hosts”, on page 693</td>
</tr>
<tr>
<td>All hosts must be able resolve hostnames of other hosts in complex</td>
<td>See section 9.2.4.5, “Hostname Resolution”, on page 694</td>
</tr>
<tr>
<td>Filesystem must be shared, on a separate host from either server host, and provide features required for failover</td>
<td>See section 9.2.4.6, “Shared Filesystem”, on page 694</td>
</tr>
<tr>
<td>Administrator must have access to filesystem from both server hosts</td>
<td>See section 9.2.4.7, “Permission Requirements”, on page 696</td>
</tr>
<tr>
<td>Same version of PBS for all components</td>
<td>See section 9.2.4.8, “Same PBS Versions Everywhere”, on page 697</td>
</tr>
<tr>
<td>Primary server’s scheduler must be able to run when primary server runs</td>
<td>See section 9.2.4.9, “Requirement for Primary Server’s Scheduler”, on page 697</td>
</tr>
<tr>
<td>Data service user account must be the same on both primary and secondary server hosts</td>
<td>See section 9.2.4.10, “Same Data Service Account on Both Server Hosts”, on page 697</td>
</tr>
<tr>
<td>Data service host must be default</td>
<td>See section 9.2.4.11, “Data Service Host Configuration Requirement”, on page 697</td>
</tr>
</tbody>
</table>
9.2.4.2 Server Host Requirements

The primary and secondary servers must run on two separate host machines. Both host machines must have the same architecture. They must be binary compatible, including word length, byte order, and padding within structures. There must be exactly one primary and one secondary server.

On an SGI ICE, use two different service nodes to run the primary and secondary servers.

9.2.4.3 Requirements for MoMs on Server Hosts

You can run a MoM on both the primary and secondary server hosts, but this is not recommended.

If a MoM is to run on both server hosts, the two MoMs must not share the same PBS_HOME/mom_priv directory. In addition, it is strongly recommended that the following be true:

- The mom_priv directory structure be replicated on a local, non-shared, filesystem. On Windows, MoM already has a local directory on each server host. On UNIX/Linux, you

### Table 9-2: Prerequisites for Failover

<table>
<thead>
<tr>
<th>Prerequisite</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>User names must be consistent across primary &amp; secondary servers hosts</td>
<td>See section 9.2.4.12, “Consistent User Names”, on page 697</td>
</tr>
<tr>
<td>The mail_from server attribute specifies an email address that is monitored. Not required, but recommended.</td>
<td>See section 9.2.4.13, “Server Mail is Monitored”, on page 698</td>
</tr>
</tbody>
</table>
must create these.

Replicate the mom_priv and mom_logs directory structures on the primary server’s host if they don’t exist there already. You must put these in the same location. Do the following on the primary server host:

```
scp -r <existing PBS_HOME/mom_priv> <local PBS_HOME/mom_priv>
scp -r <existing PBS_HOME/mom_logs> <local PBS_HOME/mom_logs>
```

Replicate the mom_priv and mom_logs directory structures on the secondary server’s host if they don’t exist there already. You must put these in the same location. Do the following on the secondary server host:

```
scp -r <existing PBS_HOME/mom_priv> <local PBS_HOME/mom_priv>
scp -r <existing PBS_HOME/mom_logs> <local PBS_HOME/mom_logs>
```

- Each MoM use its own, local, mom_priv directory structure

The PBS_MOM_HOME entry in pbs.conf specifies the location that contains the mom_priv and mom_logs directories. If PBS_MOM_HOME is specified in pbs.conf, pbs_mom uses that location instead of PBS_HOME.

To prevent the MoMs from automatically using the same directory, do one of the following:

- Recommended: Specify the separate, local PBS_MOM_HOME entry in each server host’s pbs.conf file (each pbs_mom will use the location for mom_priv specified in its PBS_MOM_HOME). Give the location of the local PBS_HOME/mom_priv that you replicated on each host. You can perform this step now, or later, when editing pbs.conf on each server host, in section 9.2.5.3, “Host Configuration for Failover on UNIX/Linux”, on page 700, or section 9.2.5.4, “Host Configuration for Failover on Windows”, on page 704.

- Use the -d option when starting at least one pbs_mom to specify that they use the local, non-default locations for mom_priv

### 9.2.4.4 Communication Between Hosts

Both the primary and secondary server hosts must be able to communicate over the network with each other and all execution hosts.

Beware of dependencies on remote file systems: The $PBS_CONF environment variable must point to pbs.conf. PBS depends on the paths in pbs.conf being available when its startup script is executed. PBS will hang if a remote file access hangs, and normal privileges don’t necessarily carry over for access to remote file systems. For example, a FAT filesystem mounted via NFS won’t support permissions.
Hostname resolution must work between each host in the PBS complex. Make sure that all hosts in the complex (the primary and secondary server hosts, the file server host, and all execution and client hosts) are set up so that they can resolve the names of all other hosts in the complex. If you are not sure whether hostname resolution is working, run the pbs_hostn command at each host, testing the hostnames of the other hosts. The pbs_hostn command will return the canonical hostname of the specified host.

Shared Filesystem

The filesystem you use for the machines managed by PBS should be highly reliable. We recommend, in this order, the following filesystems:

- HA DAS
- DAS, such as xfs or gfs
- HA NFS
- NFS

PBS_HOME is the top directory used by the PBS server. The primary and secondary servers share the same PBS_HOME directory. The PBS_HOME directory must conform to the following:

- The PBS_HOME directory must be available under the same name to both the primary and secondary server hosts.
- The PBS_HOME directory must be on a file system which meets the following requirements:
  - It should reside on a different machine from either of the server hosts.
  - It must be shared by the primary and secondary server hosts.
  - It must be reliable. The file system must be always available to both the primary and secondary servers. A failure of the file system will stop PBS from working.
  - The file system protocol must provide file locking support.
  - The file locking daemons must be running.
  - For UNIX/Linux, the filesystem must support POSIX (Open Group) file semantics.
  - It must support concurrent read and write access from two hosts.
  - It must support multiple export/mounting.

If your filesystem does not conform to the specifications above, follow the steps in the next sections.
9.2.4.6.i Using NFS Filesystems

When using NFS for PBS_EXEC, NFS must be configured to allow root access and to allow setuid-root programs to execute from it.

If possible, mount NFS file systems synchronously (without caching) to avoid reliability problems.

NFS filesystems should be hard mounted.

9.2.4.6.ii Setting Up the Shared Filesystem on UNIX/Linux

You can use NFS or another filesystem protocol to set up the shared file system on which PBS_HOME resides. Examples are SGI CXFS, IBM GPFS, and Red Hat GFS. Make sure your protocol supports:

• Multiple export/mounting
• Simultaneous read/write from two hosts
• File locking support

To set up your file system:

1. Choose a machine for the file server host. This machine must not be either of the server hosts.

2. Make sure the file system is mounted by both the primary and secondary server hosts. For NFS, make sure the file system is hard mounted by both hosts.

3. Make sure the file system can provide file locking. For NFS, the lock daemon, lockd, must be running.

4. Make sure that PBS_HOME is available under the same name to both the primary and secondary server hosts.

9.2.4.6.iii Setting up a Shared PBS_HOME on Windows

Under Windows, you can use the network share facility to set up the shared PBS_HOME. The primary and secondary servers will share a PBS_HOME directory that is located on a network share file system on a non-server host.
To set up your file system:

1. Choose a machine for the file system host. This machine must not be either of the server hosts. Log on to this machine.

2. On the file system host, create a folder named, for example, C:\pbs_home.

3. Using Windows Explorer, select the C:\pbs_home folder by right-clicking it, and choose “Properties”.

4. Select the Sharing tab, and click the checkbutton that says "Share this folder"; specify "Full Control" permissions for the local Administrators group on the local computer.

5. Make sure that PBS_HOME is available under the same name to both the primary and secondary server hosts.

6. Make sure the file system can provide file locking.

7. Copy the files from the local PBS home directory (where it was put during installation) onto the shared PBS_HOME directory.

   On Windows XP:
   
   xcopy /o /e "\Program Files\PBS Pro\home" "\<shared filesystem host>\pbs_home"

   On Vista, 32-bit:

   robocopy "\Program Files\PBS Pro\home" "\<shared filesystem host>\pbs_home" /R

   On Vista, 64-bit:

   robocopy "\Program Files (x86)\PBS Pro\home" "\<shared filesystem host>\pbs_home" /R

   Make sure that the copy of each file is successful. For any files that failed to be copied, perform the copy by hand.

9.2.4.7 Permission Requirements

The PBS_HOME directory must meet the security requirements of PBS. Each parent directory above PBS_HOME must be owned by root and writable by root only.

The PBS_HOME directory must be readable and writable from both server hosts by root (Administrator) on UNIX/Linux.
On Windows, PBS_HOME must have Full Control permissions for the local "Administrators" group on the local host. Example:

```
Z:\ BUILTIN\Administrators:F
<domain>\<PBS service account>:F
BUILTIN\Administrators:F
Everyone:R
```

### 9.2.4.8 Same PBS Versions Everywhere

Both server hosts, all the execution hosts, and all the client hosts must run the same version of PBS Professional.

### 9.2.4.9 Requirement for Primary Server’s Scheduler

The primary scheduler must be able to run whenever the primary server is running. The primary server can use only the primary scheduler. If the primary server becomes active but cannot use its own scheduler, PBS will not be able to schedule jobs.

### 9.2.4.10 Same Data Service Account on Both Server Hosts

The data service account must be the same on both server hosts. The UID of the data service account must be identical on both the primary and secondary server hosts. We recommend that the data service account is called `pbsdata`.

If you change either service account, both must be changed at the same time and both servers must be restarted.

### 9.2.4.11 Data Service Host Configuration Requirement

The DATA_SERVICE_HOST parameter must not be set in `pbs.conf`. If this parameter is set, failover cannot take place.

### 9.2.4.12 Consistent User Names

User names must be consistent across the primary and secondary server hosts. If usernames are not consistent, jobs are killed.
9.2.4.13 Server Mail is Monitored

Use the `qmgr` command to set the `mail_from` server attribute to an address that is monitored regularly:

```
qmgr: s server mail_from=<address>
```

9.2.5 Configuring Failover

9.2.5.1 Overview of Configuring Failover

If PBS is not already installed, install it according to the PBS Professional Installation & Upgrade Guide.

Please make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 691.

Make a copy of your PBS configuration. Follow the instructions in “Back Everything Up” on page 134 in the PBS Professional Installation & Upgrade Guide.

The following table contains a guide to the steps in configuring PBS for failover for both UNIX/Linux and Windows. The table contains a link to the description of each step.

<table>
<thead>
<tr>
<th>Step</th>
<th>UNIX/Linux</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure <code>/etc/pbs.conf</code> on each host in the complex</td>
<td>See section 9.2.5.2, “Configuring the pbs.conf File”, on page 699</td>
<td></td>
</tr>
<tr>
<td>Configure the primary server</td>
<td>See section 9.2.5.3.1, “Configuring Failover For the Primary Server on UNIX/Linux”, on page 701</td>
<td>See section 9.2.5.4.1, “Configuring Failover for the Primary Server on Windows”, on page 705</td>
</tr>
<tr>
<td>Configure the secondary server</td>
<td>See section 9.2.5.3.2, “Configuring Failover For the Secondary Server on UNIX/Linux”, on page 703</td>
<td>See section 9.2.5.4.2, “Configuring Failover for the Secondary Server on Windows”, on page 706</td>
</tr>
</tbody>
</table>
9.2.5.2 Configuring the pbs.conf File

The $PBS_CONF environment variable contains the path to the pbs.conf file. Each host in the complex must have a properly configured /etc/pbs.conf file. This file specifies the hostnames of the primary and secondary servers, the location of PBS_HOME and PBS_MOM_HOME, and whether to start a server, a scheduler, or a MoM on this host.

The name used for the server in the PBS_SERVER variable in the pbs.conf file must not be longer than 64 characters. If the short name for the server resolves to the correct host, you can use this in pbs.conf as the value of PBS_SERVER. However, if the fully-qualified domain name is required in order to resolve to the correct host, then the this must be the value of the PBS_SERVER variable.

### Table 9-4: Parameters in pbs.conf for Failover

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS_EXEC</td>
<td>Path</td>
<td>Location of PBS bin and sbin directories</td>
</tr>
<tr>
<td>PBS_HOME</td>
<td>Path</td>
<td>Location of PBS working directories in shared filesystem; use specific path on that host</td>
</tr>
<tr>
<td>PBS_MOM_HOME</td>
<td>Path</td>
<td>Location of mom_priv on each host; overrides PBS_HOME for mom_priv</td>
</tr>
</tbody>
</table>
9.2.5.2.i Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

9.2.5.3 Host Configuration for Failover on UNIX/Linux

- Make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 691.
- PBS should already be installed in the default location on the primary and secondary server hosts and on the execution hosts. The client commands should already be installed on the client hosts.
- If the primary server and scheduler are running, shut them down. See “qterm” on page 231 of the PBS Professional Reference Guide.

Table 9-4: Parameters in pbs.conf for Failover

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS_PRIMARY</td>
<td>FQDN of hostname</td>
<td>Hostname of primary server host</td>
</tr>
<tr>
<td>PBS_SECONDARY</td>
<td>FQDN of hostname</td>
<td>Hostname of secondary server host</td>
</tr>
<tr>
<td>PBS_SERVER</td>
<td>Hostname</td>
<td>Fully-qualified domain name of primary server host</td>
</tr>
<tr>
<td>PBS_START_MOM</td>
<td>0 or 1</td>
<td>Specifies whether a MoM is to run on this host</td>
</tr>
<tr>
<td>PBS_START_SCHED</td>
<td>0 or 1</td>
<td>Specifies whether scheduler is to run on this host</td>
</tr>
<tr>
<td>PBS_START_SERVER</td>
<td>0 or 1</td>
<td>Specifies whether server is to run on this host</td>
</tr>
</tbody>
</table>
9.2.5.3.3 Configuring Failover For the Primary Server on UNIX/Linux

1. Make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 691.

2. Stop PBS on both the primary and secondary server hosts:
   - On the primary server’s host:
     ```bash
     <path to init.d>/init.d/pbs stop
     ```
   - On the secondary server’s host:
     ```bash
     <path to init.d>/init.d/pbs stop
     ```

3. On the primary server’s host, edit the `/etc/pbs.conf` file so that it DOES NOT include failover settings. It should look like this:
   ```
   PBS_SERVER=<short name for primary host>
   PBS_HOME=<shared location of PBS_HOME>
   PBS_START_SCHED=1
   ```
   It is recommended not to run a MoM on both server hosts. The following setting in `pbs.conf` will prevent a MoM from running:
   ```
   PBS_START_MOM=0
   ```
   If you will run a MoM on the server hosts, specify this:
   ```
   PBS_START_MOM=1
   ```
   If you will run a MoM on both server hosts, specify `PBS_MOM_HOME` on this host. The location you specify is the directory that you replicated in section 9.2.4.3, “Requirements for MoMs on Server Hosts”, on page 692:
   ```
   PBS_MOM_HOME=<location of local, replicated mom_priv>
   ```

4. On the primary server’s host, start the primary PBS server and scheduler daemons:
   ```bash
   <path to init.d>/init.d/pbs start
   ```

5. Stop the PBS server on the primary server’s host:
   ```bash
   <path to init.d>/init.d/pbs stop
   ```

6. On the primary server’s host, edit the `/etc/pbs.conf` file to include the failover settings for PBS_PRIMARY and PBS_SECONDARY. It should look like this:
   ```
   PBS_PRIMARY=<primary_host>
   PBS_SECONDARY=<secondary_host>
   PBS_SERVER=<short name for primary host>
   PBS_HOME=<shared location of PBS_HOME>
   ```
Chapter 9  Making Your Site More Robust

The primary scheduler will start automatically:

PBS_START_SCHED=1

It is recommended not to run a MoM on both server hosts. The following setting in pbs.conf will prevent a MoM from running:

PBS_START_MOM=0

If you will run a MoM on the server hosts, specify this:

PBS_START_MOM=1

If you will run a MoM on both server hosts, specify PBS_MOM_HOME on this host. The location you specify is the directory that you replicated in section 9.2.4.3, “Requirements for MoMs on Server Hosts”, on page 692:

PBS_MOM_HOME=<location of local, replicated mom_priv>

7. On the primary server’s host, start the primary PBS server and scheduler daemons:

<path to init.d>/init.d/pbs start
9.2.5.3.ii Configuring Failover For the Secondary Server on UNIX/Linux

1. Make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 691.

2. On the secondary server’s host, edit the /etc/pbs.conf file to include the following settings:
   
   PBS_PRIMARY=<primary_host>
   PBS_SECONDARY=<secondary_host>
   PBS_SERVER=<short name for primary host>
   PBS_HOME=<shared location of PBS_HOME>

   The secondary server will start its own scheduler if it needs to; a scheduler should not automatically start on the secondary server host. Include the following so that a scheduler does not automatically start on this host:
   
   PBS_START_SCHED=0

   It is recommended not to run a MoM on both server hosts. The following setting in pbs.conf will prevent a MoM from running:
   
   PBS_START_MOM=0
   
   If you will run a MoM on the server hosts, specify this:
   
   PBS_START_MOM=1
   
   If you will run a MoM on both server hosts, specify PBS_MOM_HOME on this host. The location you specify is the directory that you replicated in section 9.2.4.3, “Requirements for MoMs on Server Hosts”, on page 692:
   
   PBS_MOM_HOME=<location of local, replicated mom_priv>

3. On the secondary server’s host, to change the delay time between failure of the primary server and activation of the secondary server from its default of 30 seconds, use the -F <delay> option on the secondary server's command line in the PBS start script on the secondary server’s host. Edit the init.d/pbs script so that the server is invoked with the -F <delay> option:
   
   pbs_server -F <delay>

   See “pbs_server” on page 89 of the PBS Professional Reference Guide.

4. On the secondary server’s host, start the secondary PBS server daemon:
   
   <path to init.d>/init.d/pbs start
Chapter 9  Making Your Site More Robust

9.2.5.3.iii  Configuring Failover For Execution and Client Hosts on UNIX/Linux

1. Make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 691.

2. On each execution or client host, configure the /etc/pbs.conf file to include the following parameters:

   PBS_PRIMARY=<primary_host>
   PBS_SECONDARY=<secondary_host>
   PBS_SERVER=<short name for primary host>
   PBS_HOME=<location of PBS HOME>

   The pbs.conf files on execution hosts are already configured to start the MoM daemon only. Similarly, the pbs.conf files on client hosts are already configured to start no daemons.

3. On each execution host, restart the MoM:

   <path to init.d>/init.d/pbs start

9.2.5.4  Host Configuration for Failover on Windows

- PBS should already be installed in the default location on the primary and secondary server hosts and on the execution and client hosts. See the PBS Professional Installation and Upgrade Guide for instructions.

- Please make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 691.

- Configure server failover from the console of each host or through VNC. Do not use Remote Desktop. Setting up the server failover feature from a Remote Desktop will cause problems. In particular, starting the server on either the primary or secondary server host will lead to the error:

   error 1056: Service already running

   This will happen even though PBS_HOME\server_priv\server.lock and PBS_HOME\server_priv\server.lock.secondary files do not exist.

- Stop all the PBS services on the primary and secondary server hosts. On each host:

   net stop pbs_server
   net stop pbs_mom
   net stop pbs_sched
   net stop pbs_rshd
9.2.5.4.i Configuring Failover for the Primary Server on Windows

1. Make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 691.

2. On the primary server host, specify the location of PBS_HOME for the primary server:
   
   ```bash
   pbs-config-add "PBS_HOME=\<shared filesystem host>\pbs_home"
   ```

3. On the primary server host, specify the primary and secondary server names in the pbs.conf file by running the following commands:

   ```bash
   pbs-config-add "PBS_SERVER=<short name of primary server host>"
   pbs-config-add "PBS_PRIMARY=<FQDN of primary server host>"
   pbs-config-add "PBS_SECONDARY=<FQDN of secondary server host>"
   ```

4. On the primary server host, specify that the primary server host will run a server:

   ```bash
   pbs-config-add "PBS_START_SERVER=1"
   ```

5. On the primary server host, specify that the primary scheduler will automatically start:

   ```bash
   pbs-config-add "PBS_START_SCHED=1"
   ```

6. It is recommended not to run a MoM on both server hosts. The following setting in pbs.conf will prevent a MoM from running. On the primary server’s host:

   ```bash
   pbs-config-add "PBS_START_MOM=0"
   ```

   If you will run a MoM on the primary server’s host, indicate that a MoM should run, and specify the local PBS_MOM_HOME by running the following commands:

   ```bash
   pbs-config-add "PBS_START_MOM=1"
   pbs-config-add "PBS_MOM_HOME=C:INSTALLATION_BASE\home"
   ```

7. If you are going to run a MoM on this host, start it:

   ```bash
   net start pbs_mom
   ```

8. Now start the PBS server, scheduler and rshd on this host:

   ```bash
   net start pbs_server
   net start pbs_sched
   net start pbs_rshd
   ```

9. Set the managers attribute on the primary server so that when the secondary server takes over, you can still run privileged tasks under the Administrator account or from a peer pbs_server:

   ```bash
   Qmgr: set server managers="<account that installed PBS>@*,<name of PBS service account>@*"
   ```
Configuring Failover for the Secondary Server on Windows

1. Make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 691.

2. On the secondary server host, specify the location of PBS_HOME for the secondary server:
   ```shell
   pbs-config-add "PBS_HOME=\<shared filesystem host>\pbs_home"
   ```

3. On the secondary server host, specify the primary and secondary server names in the pbs.conf file by running the following commands:
   ```shell
   pbs-config-add "PBS_SERVER=<short name of primary server host>"
   pbs-config-add "PBS_PRIMARY=<FQDN of primary server host>"
   pbs-config-add "PBS_SECONDARY=<FQDN of secondary server host>"
   ```

4. Specify that the secondary server host will run a server:
   ```shell
   pbs-config-add "PBS_START_SERVER=1"
   ```

5. On the secondary server’s host, modify pbs.conf to specify how the secondary server will start its scheduler:
   ```shell
   pbs-config-add "PBS_START_SCHED=1"
   ```

6. On the secondary server’s host, specify the startup type for the scheduler:
   Go to Control Panel->Administrative Tools->Services, and bring up the PBS_SCHED service dialog, select the General tab, and specify "Manual" for startup type.

7. It is recommended not to run a MoM on both server hosts. The following setting in pbs.conf will prevent a MoM from running. On the secondary server’s host:
   ```shell
   pbs-config-add "PBS_START_MOM=0"
   ```
   If you will run a MoM on the secondary server’s host, indicate that a MoM should run, and specify the local PBS_MOM_HOME by running the following commands:
   ```shell
   pbs-config-add "PBS_START_MOM=1"
   pbs-config-add "PBS_MOM_HOME=C:INSTALLATION_BASE\home"
   ```

8. If you are going to run a MoM on this host, start it:
   ```shell
   net start pbs_mom
   ```

9. Now start the secondary server on this host:
   ```shell
   net start pbs_server
   ```
   You will see the following message:
“PBS_SERVER could not be started”

This message appears because the secondary server is inactive, waiting for the primary server to fail.

10. On the secondary server’s host, to change the delay time between failure of the primary server and activation of the secondary server from its default of 30 seconds, use Start Menu->Control Panel->Administrative Tools->Services, choosing PBS_SERVER, and specify in the “Start Parameters” entry box:

   -F <delay>

Then restart the secondary server.

The Services dialog does not save the “Start Parameters” value for future restarts. The default delay value specified in the PBS start script, or the default if there is none, will be in effect on subsequent restarts.

9.2.5.4.iii Configuring Failover for Execution and Client Hosts on Windows

1. Make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 691.

2. On each execution or client host, specify the location of PBS_HOME for the primary server:

   pbs-config-add "PBS_HOME=\<shared filesystem host>\pbs_home"

3. On each execution or client host, specify the primary and secondary server names in the pbs.conf file by running the following commands:

   pbs-config-add "PBS_SERVER=<short name of primary server host>"
   pbs-config-add "PBS_PRIMARY=<FQDN of primary server host>"
   pbs-config-add "PBS_SECONDARY=<FQDN of secondary server host>"

4. If this is an execution host, restart the MoM:

   net start pbs_mom

9.2.6 Configuring Failover with Other PBS Features

9.2.6.1 Configuring Failover to Work with Routing Queues

You must configure failover to work with routing queues which have destinations in another complex. No additional configuration is required for routing queues which have destinations in the same complex.
Chapter 9  Making Your Site More Robust

For a routing queue in one complex which points to a queue Q1 in another PBS complex that is set up for failover, it is a good idea to specify both Q1@primary.example.com and Q1@secondary.example.com as destinations.

For example, if a routing queue has a destination queue at another complex’s primary server:

```
Qmgr: set queue r66 route_destinations=workq@primary.example.com
```

you need to add the same queue at the other complex’s secondary server:

```
Qmgr: set queue r66 route_destinations+=workq@secondary.example.com
```

See section 2.2.6, “Routing Queues”, on page 24.

9.2.6.2 Configuring Failover to Work With Peer Scheduling

For peer queueing where the furnishing complex is set up for failover:

• You must list the furnishing queue at both primary and secondary servers. If the furnishing queue is Q1, the peer_queue line in the pulling complex’s sched_config file must list Q1@primary.example.com and Q1@secondary.example.com

For peer queueing where the pulling complex is set up for failover:

• You must add <manager>@primary.example.com and <manager>@secondary.example.com to the list of managers at the furnishing server.

See section 4.8.31, “Peer Scheduling”, on page 205.

9.2.6.3 Configuring Failover to Work With Access Controls

If you are using access control on the server (the acl_host_enable server attribute is set to True and the acl_hosts server attribute is specified), add the secondary server to the host list in acl_hosts:

```
Qmgr: s server acl_hosts+=<secondary server host>
```

See section 8.3.4, “ACLs”, on page 649.

9.2.7 Using PBS with Failover Configured

9.2.7.1 Stopping Servers

To stop both servers when the primary server is active, and the secondary server is running and idle, do the following:

```
qterm -f
```
To stop the primary server and leave the secondary server idle:

```
qu_term -i
```

To stop the secondary server only:

```
qu_term -F
```

### 9.2.7.2 Starting Servers

After configuring the servers, you can start them in any order.

If you want to start the primary server when the secondary server is the active server, you do not need to stop the secondary. When the primary server starts, it informs the secondary that the secondary can become idle.

However, if there is a network outage while the primary starts and the secondary cannot contact it, the secondary will assume the primary is still down, and remain active, resulting in two active servers. In this case, stop the secondary server, and restart it when the network is working:

UNIX/Linux:

```
qu_term -F
pbs_server
```

Windows:

```
qu_term -F
net start pbs_server
```

To restart the secondary server while it is the active server:

```
pbs_server -F -1
```

The secondary server makes one attempt to contact the primary server, and becomes active immediately if it cannot.

See “pbs_server” on page 89 of the PBS Professional Reference Guide and “qterm” on page 231 of the PBS Professional Reference Guide.

### 9.2.8 Recommendations and Caveats

- **Do not** start or stop the data service using anything except the `pbs_dataservice` command. Start or stop the data service using only the `pbs_dataservice` command.
- If you do not wish for the secondary server to take over, use the `-i` option to the `qterm` command when stopping the primary server.
- When the primary server is active, and the secondary server is running and idle, the `pbs`
start/stop script stops the active server, but leaves the idle server running. This means that the idle server becomes the active server.

- PBS_HOME should not be on either server host
- Neither PBS server should be the NFS fileserver
- The primary scheduler must be able to run when the primary server is started, otherwise no jobs will be scheduled; the primary server cannot use the secondary scheduler.
- Just because servers are redundant, that doesn't mean that your complex is. Look for single points of failure.
- If the “take over” delay time specified with the pbs_server -F option is too long, there may be a period, up to that amount of time, when clients cannot connect to either server.
- If the “take over” delay time specified with the pbs_server -F option is too short and there are transient network failures, then the secondary server may attempt to take over while the primary server is still active.
- While the primary server is active and the secondary server is inactive, the secondary server will not respond to any network connection attempts. Therefore, you cannot status the secondary server to determine whether it is running.
- If you start the secondary server manually, it will not start its own scheduler. You must start the secondary server’s scheduler manually, if required.
- If the secondary server is running, and the primary server cannot contact the secondary server when the primary server is restarted, the primary assumes the secondary is not running and takes over. This can result in two servers running at once.

### 9.2.9 Troubleshooting Failover

#### 9.2.9.1 PBS Won’t Start

- If you see the following error:
  
  "Failover is configured. Temporarily disable failover before running pbs_ds_password"
  
  This means that PBS was started for the first time with failover configured. PBS cannot be started for the first time with failover configured. Remove definitions for PBS_PRIMARY and PBS_SECONDARY from pbs.conf on the primary server’s host, start PBS, stop PBS, replace the definitions, and start PBS again.
9.2.9.2 Primary and Secondary Servers Both Running

If both servers are running, this may be because the primary server was stopped and then restarted, and while the primary was stopped, the secondary began to take over. While the secondary server was coming up, it was not able to receive the message from the primary server indicating that it should go idle, or it couldn’t register with the primary.

To avoid this problem, use the -i option to the qterm command, which tells the secondary server to remain idle.

9.2.9.3 Primary or Secondary Server Fails to Start

It does not matter in which order the primary and secondary servers are started.

If the primary or secondary server fails to start with the error:

```
another server running
```

then check for the following conditions:

1. There may be lock files left in PBS_HOME/server_priv that need to be removed.
   - The primary and secondary servers use different lock files:
     - primary: server.lock
     - secondary: server.lock.secondary

2. On UNIX, the RPC lockd daemon may not be running. You can manually start this daemon by running as root:

   ```
   <path to daemon>/rpc.lockd
   ```

   Check that all daemons required by your NFS are running.

9.2.9.4 Windows: Primary Server Fails to Resume Control

If the primary server fails to take over for the secondary server when it should, reboot the primary server’s host.

9.3 Checkpoint and Restart

PBS Professional allows you to configure MoM to checkpoint jobs using your scripts and checkpoint tools. In addition, users may manage their own checkpointing from within their application.
9.3.1 Terminology

Application Checkpoint
The application performs its own checkpointing when it receives the appropriate signal etc.

Checkpoint and Abort, checkpoint_abort
The checkpoint script or tool writes a restart file, then PBS kills and requeues the job. The job uses the restart file when it resumes execution.

Restart
A job that was stopped after being checkpointed while previously executing is executed again, starting from the point where it was checkpointed.

Restart File
The job-specific file that is written by the checkpoint script or tool. This file contains any information needed to restart the job from where it was when it was checkpointed.

Restart Script
The script that MoM runs to restart a job. This script is common to all jobs, and so must use the information in a job’s restart file to restart the job.

Snapshot Checkpoint
The checkpoint script or tool writes a restart file, and the job continues to execute. The job resumes based on this restart file if the system experiences a problem during the job’s subsequent execution.

9.3.2 How Checkpointing Works

When a job is checkpointed, MoM executes a checkpoint script. The checkpoint script saves all of the information necessary to checkpoint the job. If the checkpoint is for a snapshot, the job continues to run. If the job is checkpointed and aborted, PBS kills and requeues the job after checkpointing it.

When a job is restarted, MoM executes a restart script. The restart script uses the saved information to restore the job. The manner of restarting the job depends on how it was checkpointed:

• If the job was checkpointed during shutdown, the job becomes eligible to run when PBS is restarted, and will start from where it was checkpointed.
• If the job was checkpointed by the scheduler because it was preempted, the scheduler briefly applies a hold, but releases the hold immediately after checkpointing the job, and
runs the restart script when the job is scheduled to run.

- If the job was checkpointed and held via the `qhold` command, the hold must be released via the `qrls` command for the job to be eligible to run. Then when the scheduler next runs the job, the restart script is executed, and the job runs from where it was checkpointed.

A restarted job always runs on the same machine where it was running when it was checkpointed. On Cray systems, a job will run on the original login node, but job processes are not guaranteed to run on the same compute nodes as before the checkpoint.

You can configure PBS to requeue jobs that were snapshot checkpointed while they ran, if the epilogue exits with a special value. These jobs are then restarted from the restart file.

You can provide checkpointing for jobs using any combination of scripts that you write and third-party checkpointing tools such as Meiosys Checkpoint and BLCR (Berkeley Lab Checkpoint/Restart). You can configure PBS to trigger the scripts or tools, so that the scripts and/or tools create a job’s restart file.

You can configure one behavior for snapshots, and another behavior for checkpoint and abort. Some applications provide their own checkpointing, which is triggered, for example, when the application receives a signal or detects a change in a file.

### 9.3.2.1 Types of Checkpointing

#### 9.3.2.1.i Checkpoint and Abort

Checkpoint and abort is used when a job is checkpointed before being killed. When the job is checkpointed, the following takes place:

- MoM runs the `checkpoint_abort` script; the checkpoint script or tool writes a restart file specific to that job
- The `checkpoint_abort` script terminates the job
- PBS requeues the job
- If the job was held via the `qhold` command, PBS applies a hold to the job (puts it in the `Held` state)

The job resumes execution based on the information in the restart file.

Checkpoint and abort is applied when:

- The `qhold` command is used on a job
- The server is shut down via `qterm -t immediate` or `qterm -t delay`
- The scheduler preempts a job using the `checkpoint` method
9.3.2.1.ii  Snapshot Checkpoint

Snapshot checkpointing is used for checkpointing a job at regular intervals. The job continues to run. When the job is checkpointed, the following takes place:

- MoM runs the snapshot checkpoint script; the checkpoint script or tool writes a restart file specific to that job
- The job continues to execute

The job resumes execution based on this restart file if the system crashes or if the epilogue returns -2. See section 9.3.7.3, “Requeueing via Epilogue”, on page 730.

The interval can be specified by the user via `qsub -c <checkpoint spec>`. You can specify a default interval, in the `checkpoint_min` queue attribute, or in the `Checkpoint job` attribute. See “`qsub`” on page 211 of the PBS Professional Reference Guide and “Job Attributes” on page 375 of the PBS Professional Reference Guide.

9.3.2.1.iii  Application Checkpoint

Application checkpointing is when an application checkpoints itself. PBS can be used to trigger application checkpointing, but does not manage the checkpoint files or process. Application checkpointing can be triggered when the application receives a signal or detects a change in a file.

9.3.2.2  Events That Trigger Checkpointing

The following table lists the events that can trigger checkpointing, and the kind of checkpointing that is used.

<table>
<thead>
<tr>
<th>Event</th>
<th>Type of Checkpointing Used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <code>qhold</code> command is used on a job</td>
<td>checkpoint_abort</td>
<td>See section 9.3.7.6, “Holding a Job”, on page 731</td>
</tr>
<tr>
<td>Server shut down via <code>qterm -t immediate or qterm -t delay</code></td>
<td>checkpoint_abort</td>
<td>See section 9.3.7.2, “Checkpointing During Shutdown”, on page 729</td>
</tr>
</tbody>
</table>
9.3.2.3 Effect of Checkpointing on Jobs

When a job is checkpointed and aborted (requeued), its accumulated queue waiting time depends on how that time is calculated:

- If you are using eligible time, the accumulated waiting time is preserved
- If you are not using eligible time, the accumulated waiting time is lost

The job exit code for being checkpointed and aborted is -12, named \texttt{JOB EXEC CHKP}.

When a job is restarted, it runs on the same machine as it did when it was checkpointed. On Cray machines, the job runs on the original login node, but may run on different compute nodes.

9.3.2.4 Effect of Checkpointing on Job Resources

When a job is checkpointed and aborted, all of its resources are freed.

A snapshot checkpoint does not affect a job’s resources.
9.3.2.5 Restarting a Job

When a job is restarted, MoM runs the restart script specified in the $action restart MoM parameter. This script looks in the checkpoint directory (see section 9.3.6.5, “Specifying Checkpoint Path”, on page 727) for the restart file for that job. It uses the information in that file to restart the job.

For a job that was checkpointed and aborted because it was held, the job has had a hold placed on it so that it will not be eligible for execution until the hold is released. In order for a checkpointed and held job to be eligible for execution, the hold must be removed using the qr1s command. The job’s owner can remove a User hold, but other holds must be removed by a Manager or Operator. See “qr1s” on page 179 of the PBS Professional Reference Guide.

If the job was preempted via checkpointing, the scheduler releases the hold on the job immediately after checkpointing the job. This will show up in the scheduler’s log file, but the job will not appear to be held because the hold duration is very short.

A job that was checkpointed and requeued during shutdown is not held. This job is eligible for execution as soon as the necessary daemons are back up. See section 9.3.7.4, “Checkpointed Jobs and Server Restart”, on page 730.

A job that was snapshot checkpointed and later requeued because the epilogue returned a special exit status is requeued in the Q state, and is eligible to be restarted when the scheduler selects it for execution.

When a checkpointed and aborted job is restarted, MoM resumes tracking the job. She tracks either the original PID of the job, or the PID of the restart script, depending on the setting of the $restart_transmogrify MoM parameter. See section 9.3.4.3, “Setting $restart_transmogrify MoM Parameter”, on page 719.

9.3.3 Prerequisites for Checkpointing Jobs

The following are the prerequisites for checkpointing jobs:

- The MoM must be configured for checkpointing
  - Specified checkpoint directories must correspond to available directories (see section 9.3.6.5, “Specifying Checkpoint Path”, on page 727)
  - Checkpoint and restart MoM configuration parameters must be specified (see section 9.3.4.2, “Specifying Checkpoint and Restart Parameters”, on page 718)
- A checkpointing script or tool must be available for each type of checkpointing to be used
9.3.3.1 Restrictions on Checkpointing

- Checkpointing is not supported for job arrays.
- PBS does not directly support OS-level checkpointing.
- You can configure only one snapshot script, so if more than one kind of snapshot checkpointing is required, the script must distinguish which kind of snapshot to perform.
- You can configure only one checkpoint_abort script, so if more than one kind of checkpoint_abort is required, the script must also distinguish which kind of checkpoint_abort to perform.
- You can configure only one restart script. The restart script is run once for each of the job’s tasks, so if some restarts are for application checkpointing, the script must handle those restarts correctly (application restarts may require only one iteration.)
- A restarted job must run on the same machine where it was running when it was checkpointed. Note that on Cray systems, the job runs on the original login node, but may run on different compute nodes.
- Checkpointing cannot be used for interactive jobs. See section 9.3.8, “Advice and Caveats”, on page 732.

9.3.4 Configuring Checkpointing

9.3.4.1 Overview of Configuring Checkpointing

You configure checkpointing by editing the MoM configuration file, PBS_HOME/mom_priv/config. You edit MoM configuration parameters to do the following:

- Specify script paths
  - Specify path to checkpoint_abort script, if needed
  - Specify path to snapshot script, if needed
  - Specify path to restart script
- Set $restart_transmogrify MoM parameter to fit your restart script
- Make the checkpoint path match that specified in the restart script

9.3.4.1.i Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.
9.3.4.2 Specifying Checkpoint and Restart Parameters

To configure checkpointing, you specify a path to a script that MoM executes when checkpointing is called for. You can specify a separate path/script for each of checkpoint_abort, snapshot, and restart using the following MoM configuration parameters:

$\text{action checkpoint timeout !path/script script-args}$

Specifies snapshot behavior.

$\text{action checkpoint_abort timeout !path/script script-args}$

Specifies checkpoint_abort behavior.

$\text{action restart timeout !path/script script-args}$

Specifies restart behavior.

where

$\text{action}$

Specifies that MoM perform the indicated action.

checkpoint

MoM executes the script specified in path/script once for each of the job’s tasks when a snapshot is called for.

checkpoint_abort

MoM executes the script specified in path/script once for each of the job’s tasks when a checkpoint_abort is called for.

restart

MoM executes the script specified in path/script once for each of the job’s tasks when a restart is called for.

timeout

The number of seconds allowed for the script or tool to execute. The value of the $\text{restart_transmogrify}$ MoM parameter determines whether this limit is applied. Values for $\text{restart_transmogrify}$, and resulting behavior:

$False$

If the script/tool does not finish running during this time, it is killed and handled as if it had returned failure.

$True$

No timeout limit is applied.

path/script

The path to the script, including the name of the script. The path can be absolute or relative. If the path is relative, it is relative to PBS_HOME/mom_priv.
Examples of absolute paths and script names:

`/usr/bin/checkpoint/snapshot`
`/usr/bin/checkpoint/checkpt-abort`
`/usr/bin/checkpoint/restart`

### script-args

These are the arguments to the script, if any.

PBS automatically expands some arguments to checkpoint and restart scripts. The following table lists the arguments that are expanded by PBS:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%globid</code></td>
<td>Global ID (no longer used)</td>
</tr>
<tr>
<td><code>%jobid</code></td>
<td>Job ID</td>
</tr>
<tr>
<td><code>%sid</code></td>
<td>Session ID</td>
</tr>
<tr>
<td><code>%taskid</code></td>
<td>Task ID</td>
</tr>
<tr>
<td><code>%path</code></td>
<td>File or directory name to contain restart files</td>
</tr>
</tbody>
</table>

#### 9.3.4.2.i Examples of Checkpoint and Restart Parameters

The following are examples of snapshot, checkpoint_abort, and restart MoM parameters:

```
$action checkpoint 60 !/usr/bin/checkpoint/snapshot %jobid %sid %taskid %path
$action checkpoint_abort 60 !/usr/bin/checkpoint/checkpt-abort %jobid %sid %taskid %path
$action restart 30 !/usr/bin/checkpoint/restart %jobid %sid %taskid %path
```

#### 9.3.4.3 Setting $restart_transmogrify MoM Parameter

The `$restart_transmogrify` MoM parameter controls how MoM runs the restart script, and whether she expects to resume tracking the job’s original PID or a new PID. When she runs a restart script, MoM forks a child process, which `exec()`s the start script. If `$restart_transmogrify` is `True`, the start script becomes the top task of the job. If `$restart_transmogrify` is `False`, the start script does not become the top task of the job.
Chapter 9  Making Your Site More Robust

If your restart script preserves the job’s original PID, set \$restart_transmogrify to False. This way, the script does not become the top task of the job, and MoM continues to track the job’s original PID.

If your restart script results in a new PID for the job, set \$restart_transmogrify to True. This way, the restart script becomes the top task of the job, and MoM tracks the PID of the new top process, which is the script.

9.3.4.4  Checkpointing on Cray Systems

Checkpointing on the Cray XT uses BLCR. Use the PBS generic checkpoint facility for checkpoint and restart. Write checkpoint and restart scripts that use BLCR commands.

9.3.4.4.i  Cray Checkpointing Requirements

• CPR jobs must be linked with the Cray MPT 3.0.1 or later libraries, which have integrated BLCR support. The application must be an MPT application (either MPI or shmem) and linked with the BLCR libcr. Most users will simply need to load the BLCR module via the following:

  module load blcr

  and compile their application in the usual way for MPI or shmem applications.

• The Cray XT system must have the BLCR kernel module installed and loaded.

• For the Lustre file system, ensure the directory to which the context files will be written has a stripe count of 1. For example, if the checkpoint path is /scratch/
BLCR_checkpoint_dir, do the following:

  # cd /scratch
  # lfs setstripe -c 1 BLCR_checkpoint_dir

• Ensure that the context file's destination is in the shared filesystem. The context file is written in the directory where the cr_checkpoint command is performed.

• Cray CPR must be enabled on the login nodes. As root, use xtopview from the boot node, and do the following:

  boot# xtopview -x /etc/node_classes -c login
  login> chkconfig blcr on

• Environment variables must be set correctly on the login node. The file PBS_HOME/pbs_environment, which is available to user job scripts, must contain the following:

  LD_PRELOAD=/usr/lib64/libcr_run.so.0
  LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib:/usr/lib64

• Make sure that the ALPS checkpoint-restart rpm is installed on the root that is being used by those running jobs, or set the CRAY_ROOTFS environment variable to point to the
Making Your Site More Robust

Chapter 9

root where the checkpoint-restart rpm is installed:

On a CNL system the rpm might be called "cray-alpscpr-cnl".

If users are using the DSL root and the *alpscpr* rpm is installed in the DSL root, checkpoint-restart works without having to set the CRAY_ROOTFS environment variable.

When the *alpscpr* rpm is installed in a different root from the one being used by users, set the CRAY_ROOTFS environment variable to point to the root where it is installed.

For example, if users are using the DSL root and the *alpscpr* rpm is installed on the INITRAMFS root, set it as shown:

```
CRAY_ROOTFS=INITRAMFS
```

9.3.4.4.ii Cray Checkpointing Caveats

- On Cray systems, a job will run on the original login node, but job processes are not guaranteed to run on the same compute nodes as before the checkpoint.
- Do not use periodic checkpointing based on CPU time (via `qsub -c c`) for MPT jobs on a Cray. While the MoM runs and tracks CPU usage on the login nodes, the application is run on one or more compute nodes, where CPU usage by the compute node cannot be tracked by the MoM.

9.3.5 Parameters and Attributes Affecting Checkpointing

9.3.5.1 MoM Configuration Parameters Affecting Checkpointing

```
$action checkpoint <timeout> !<script-path> <args>
Checkpoints the job, allowing the job to continue running.

$action checkpoint_abort <timeout> !<script-path> <args>
Checkpoints, kills, and requeues the job.

$action restart <timeout> !<script-path> <args>
Restarts checkpointed job.

The <timeout> is the time allowed for checkpoint or restart script to run.

$checkpoint_path <path>
MoM passes this parameter to the checkpoint and restart scripts. This path can be absolute or relative to PBS_HOME/mom_priv. Overrides default. Overridden by path specified in the pbs_mom -c option and by PBS_CHECKPOINT_PATH environment variable.
```
$\text{restart\_background} < \text{True|False}>

Specifies whether MoM runs the restart script in the background (MoM doesn’t wait) or foreground (MoM waits). When set to True, MoM runs the restart script in the background.

Automatically set by MoM; Controlled by value of $\text{restart\_transmogrify}$. When $\text{restart\_transmogrify}$ is True, $\text{restart\_background}$ is set to False. When $\text{restart\_transmogrify}$ is False, $\text{restart\_background}$ is set to True.

Format: Boolean
Default: False

$\text{restart\_transmogrify} < \text{True|False}>

Specifies which PID MoM tracks for a job that has been checkpointed and restarted.

When this parameter is set to True, MoM tracks the PID of the restart script. When this parameter is set to False, MoM tracks the PID of the original job.

The value of $\text{restart\_transmogrify}$ controls the value of $\text{restart\_background}$.

Format: Boolean
Default: False

### 9.3.5.2 Options to pbs\_mom Affecting Checkpointing

- \text{C checkpoint\_directory}
  
  Specifies the path to the directory where MoM creates job-specific subdirectories used to hold each job’s restart files. MoM passes this path to checkpoint and restart scripts. Overrides other checkpoint path specification methods. Any directory specified with the \text{-C} option must be owned, readable, writable, and executable by root only (\text{rwx,---,---}, or \text{0700}), to protect the security of the restart files. See the \text{-d} option to pbs\_mom.

  Format: String
  Default: PBS\_HOME/checkpoint

### 9.3.5.3 Job Attribute Affecting Checkpointing

Checkpoint

Determines when the job will be checkpointed. Can take on one of the following values:
Making Your Site More Robust

Chapter 9

c
Checkpoint at intervals, measured in CPU time, set on the job’s execution queue. If there is no interval set on the queue, the job is not checkpointed.

\[ c=<\text{minutes of CPU time}> \]
Checkpoint at intervals of the specified number of minutes of job CPU time. This value must be greater than zero. If the interval specified is less than that set on the job’s execution queue, the queue's interval is used.

Format: Integer

w
Checkpoint at intervals, measured in walltime, set on the job’s execution queue. If there is no interval set at the queue, the job is not checkpointed.

\[ w=<\text{minutes of walltime}> \]
Checkpoint at intervals of the specified number of minutes of job walltime. This value must be greater than zero. If the interval specified is less than that set on the execution queue in which the job resides, the queue's interval is used.

Format: Integer

n
No checkpointing.

s
Checkpoint only when the server is shut down.

u
Unset. Defaults to behavior when interval argument is set to \( s \).

Default: u.

Format: String

9.3.5.4 Queue Attribute Affecting Checkpointing

checkpoint_min
Specifies the minimum number of minutes of CPU time or walltime allowed between checkpoints of a job. If a user specifies a time less than this value, this value is used instead. The value given in \( \text{checkpoint}\_\text{min} \) is used for both CPU minutes and walltime minutes. See the Checkpoint job attribute.

Format: Integer

Default: None
9.3.5.5 Environment Variable Affecting Checkpointing

PBS_CHECKPOINT_PATH
MoM passes this path to the checkpoint and restart scripts. Overridden by -C option to pbs_mom; overrides $checkpoint_path MoM parameter and default. See section 9.3.6.5, “Specifying Checkpoint Path”, on page 727.

9.3.5.6 The Epilogue

PBS will requeue a job which was snapshot checkpointed, if the epilogue returns the value 2. See section 9.3.7.3, “Requeueing via Epilogue”, on page 730.

9.3.6 Checkpoint and Restart Scripts

The restart script is run by the same MoM that ran the checkpoint script. The checkpoint and restart scripts are run for each task of the job. When MoM executes a checkpoint or restart script, she forks a child process, which exec()s the script. The restart script looks for the restart file in the job-specific subdirectory created by MoM, under the specified path. See section 9.3.6.5, “Specifying Checkpoint Path”, on page 727.

9.3.6.1 Environment Variables for Scripts

PBS sets the following variables in the checkpoint and restart scripts’ environments before running the scripts:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Value of Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>GID</td>
<td>Job owner’s group ID</td>
</tr>
<tr>
<td>HOME</td>
<td>Job owner’s PBS home directory</td>
</tr>
<tr>
<td>LOGNAME</td>
<td>Job owner’s login name</td>
</tr>
<tr>
<td>PBS_JOBCOOKIE</td>
<td>128-bit random number used as token to authenticate job processes</td>
</tr>
<tr>
<td>PBS_JOBID</td>
<td>The job’s ID</td>
</tr>
</tbody>
</table>
The Checkpoint Script

The checkpoint script writes a restart file that is specific to the job being checkpointed. The checkpoint script must save all of the information needed to restart the job. This is the information that will be used by the restart script to restart the job. PBS runs the script for each running job task, on each vnode where a task is running.

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Value of Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS_JOBNAME</td>
<td>The job’s name</td>
</tr>
<tr>
<td>PBS_MOMPORT</td>
<td>Port number on which MoM listens for resource manager requests</td>
</tr>
<tr>
<td>PBS_NODEFILE</td>
<td>Path and filename of this job’s node file</td>
</tr>
<tr>
<td>PBS_NODENUM</td>
<td>Index into the node file; index of this vnode; starts at 0</td>
</tr>
<tr>
<td>PBS_QUEUE</td>
<td>Name of the job’s execution queue</td>
</tr>
<tr>
<td>PBS_SID</td>
<td>Session ID of task for which script is being called</td>
</tr>
<tr>
<td>PBS_TASKNUM</td>
<td>Index into task table for this job; index of task for which script is being called</td>
</tr>
<tr>
<td>SHELL</td>
<td>Job owner’s login shell</td>
</tr>
<tr>
<td>UID</td>
<td>Job owner’s execution ID</td>
</tr>
<tr>
<td>USER</td>
<td>Job owner’s username</td>
</tr>
<tr>
<td>USERPROFILE</td>
<td>(Windows only) Job owner’s Windows home directory</td>
</tr>
<tr>
<td>USERNAME</td>
<td>(Windows only) Job owner’s Windows username</td>
</tr>
</tbody>
</table>

9.3.6.2 The Checkpoint Script

The checkpoint script writes a restart file that is specific to the job being checkpointed. The checkpoint script must save all of the information needed to restart the job. This is the information that will be used by the restart script to restart the job. PBS runs the script for each running job task, on each vnode where a task is running.
9.3.6.2.i Requirements for Checkpoint Script

- The first line of the script must specify the shell to be used, for example:
  `#!/bin/sh`
- The script should return the following error codes:
  - *Zero* for success
  - *Non-zero* for failure
- The script should block until the checkpoint process is finished.
- The restart file and its directory should be owned by root, and writable by root only, with permission 0755.
- Under UNIX/Linux, the checkpoint script should be owned by root, and writable by root only, with permission 0755.
- Under Windows, the checkpoint script must have at least Full Control permission for the local Administrators group.
- The checkpoint script must write the restart file(s) in the location expected by the restart script. You don’t have to use the `%path` parameter passed by MoM.
- If the script is for checkpoint-abort, the script must ensure that all processes are killed, whether directly or indirectly, for example by touching a file. All job processes must exit.

9.3.6.3 The Restart Script

The restart script does only one of the following:

- Reinstates the job’s original PID, so that MoM tracks the original PID
- Becomes the new top process of the job, so that MoM tracks the PID of the script

If `$restart_transmogrify` is set to *True*, the restart script becomes the new top task for the job, and MoM begins tracking its process ID, where she was tracking the job’s original process ID. If `$restart_transmogrify` is set to *False*, MoM continues to track the original job PID.

The restart script can use `pbs_attach` to attach job processes to the original job PID, or to the script’s PID. See "`pbs_attach` on page 45 of the PBS Professional Reference Guide.".

9.3.6.3.i Caveats for Restart Script

The `pbs_attach` command is not supported under Windows.

9.3.6.3.ii Requirements for Restart Script

The restart script must handle everything required to restart the job from the information saved by the checkpoint script.

The restart script must block until the restart process is finished.
Under UNIX/Linux, the restart script should be owned by root, and writable by root only, with permission 0755.

Under Windows, the restart script must have at least Full Control permission for the local Administrators group.

### 9.3.6.3.iii  Return Values for Restart Script

The restart script must inform PBS of success or failure. It must return one of the following:

- Zero for success
- Non-zero for failure

### 9.3.6.4  Scripts for Application Checkpointing

If a user’s application can be checkpointed periodically according to walltime or CPU time, you can use the PBS snapshot checkpoint facility to trigger snapshot checkpointing by the application.

If a user’s application can be checkpointed, you can use the PBS `checkpoint_abort` facility before shutting down PBS to avoid losing intermediate results.

Some applications produce a restart file when they are sent a specific signal, or when a specific file is affected. A checkpoint script for this purpose sends the application the correct signal, or makes the correct change to the file.

Some applications only need the checkpoint and restart scripts to be run once each. In this case, the checkpoint and restart scripts should handle this requirement.

### 9.3.6.5  Specifying Checkpoint Path

When a job is checkpointed, information about the job is saved into a file. The location for this file can be any directory accessible to MoM.

The path to the checkpoint directory is composed of two parts. The first part is common to all jobs; this part can specified. The second part is a job-specific subdirectory, created by MoM for each job, under the common directory. The job’s restart file is written in this job-specific subdirectory.

The default common directory, `PBS_HOME/checkpoint`, is provided for convenience.
You can specify the filename and the path for the common directory using any of the following methods. If the first is specified, PBS uses it. If not, and the second is specified, PBS uses the second, and so on.

- The `-C` path option to the `pbs_mom` command
- The `PBS_CHECKPOINT_PATH` environment variable
- The `$checkpoint_path` MoM configuration option in `PBS_HOME/mom_priv/config`
- The default value of `PBS_HOME/checkpoint`

The job-specific subdirectory is named with the following format:

```
<job ID>.CK
```

For example, if you specify `/usr/bin/checkpoint` for the common directory, and the job's ID is `1234.host1`, the job's restart file is written under `/usr/bin/checkpoint/1234.host1.CK`.

The restart file and its directory should be owned by root, and writable by root only.

### 9.3.6.5.i Checkpoint Path Caveats

If the checkpoint file is in `PBS_HOME/checkpoint/<job ID>.CK/`, and MoM thinks that a checkpoint failed (the checkpoint script returned non-zero), she will remove the checkpoint file. If the checkpoint script puts the checkpoint file in another location, MoM does not remove the checkpoint file.

### 9.3.7 Using Checkpointing

#### 9.3.7.1 Periodic Job Checkpointing

If a job’s Checkpoint attribute is set to `c`, `c=<minutes>`, `w`, or `w=<minutes>`, the job is periodically checkpointed. The checkpoint interval is specified either in the job’s Checkpoint attribute or in the queue’s `checkpoint_min` attribute. See “Job Attributes” on page 375 of the PBS Professional Reference Guide. The job’s Checkpoint attribute is set using the `-c <interval>` option to the `qsub` command. See “qsub” on page 211 of the PBS Professional Reference Guide.

When this attribute is set, at every `<interval>` the job is checkpointed and a restart file is written, but the job keeps running.
9.3.7.2 Checkpointing During Shutdown

The effect on jobs of shutting down PBS depends on the method used to shut PBS down. When a job is checkpointed during shutdown, MoM runs the checkpoint_abort script, and PBS kills and requeues the job. PBS does not hold the job, so the job is eligible to be run again as soon as the server starts up.

If you use the `qterm` command, there are three different suboptions to the `-t` option to control whether jobs are checkpointed, requeued, or allowed to continue running.

If you use the PBS start/stop script, the script affects only the host where the script is run. Any jobs running completely or partly on that host are killed and requeued, but not checkpointed. Any jobs not running on that host are left running.

The effect of each shutdown method is described here:

**Table 9-8: Effect of Shutdown on Jobs**

<table>
<thead>
<tr>
<th>Shutdown Method</th>
<th>Effect on Checkpointable Jobs</th>
<th>Effect on Non-checkpointable Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>qterm -t quick</code></td>
<td>Continue to run</td>
<td>Continue to run</td>
</tr>
<tr>
<td><code>qterm -t delay</code></td>
<td>Checkpointed, killed, requeued, held</td>
<td>Requeued if rerunnable; continue to run if not rerunnable</td>
</tr>
<tr>
<td><code>qterm -t immediate</code></td>
<td>Checkpointed, killed, requeued, held</td>
<td>Requeued if rerunnable; deleted if not rerunnable</td>
</tr>
<tr>
<td><code>init.d/pbs stop</code></td>
<td>Any jobs running completely or partly on host where stop script is run are killed and requeued</td>
<td>Any jobs running completely or partly on host where stop script is run are killed and requeued</td>
</tr>
<tr>
<td></td>
<td>Jobs not running on host where stop script is run are left running</td>
<td>Jobs not running on host where stop script is run are left running</td>
</tr>
</tbody>
</table>

Any running subjobs of a job array are always killed and requeued when the server is shut down.
Chapter 9  Making Your Site More Robust

9.3.7.3  Requeueing via Epilogue

You can configure MoM to requeue a failed job that was snapshot checkpointed during its execution. For example, if a job terminates, but had a hardware failure during execution, PBS can requeue the job, and MoM will run the start script, which can restart the job from its restart file.

When the job is requeued via the epilogue mechanism, it is in the Q state.

9.3.7.3.i  Requirements for Requeueing via Epilogue

The following requirements must be met in order for a job to be requeued via the epilogue mechanism:

- The epilogue must return a value of 2
- The job must have been checkpointed under the control of PBS
- The MoM must be configured with a restart script in the $action restart MoM configuration parameter
- The MoM must be configured to snapshot checkpoint jobs in the $action checkpoint MoM configuration parameter
- The jobs must request checkpointing via their Checkpoint attribute. See section 9.3.7.1, “Periodic Job Checkpointing”, on page 728
- The epilogue script in PBS_HOME/mom_priv/epilogue must return the following:
  - Zero (0) for successful termination (requeue is not required)
  - Two (2) for failure (requeue is required)

9.3.7.4  Checkpointed Jobs and Server Restart

When the server is restarted using the pbs_server -t warm command or the init.d/ pbs_start script, jobs that were checkpointed and aborted upon shutdown are waiting in their queues, and are eligible to be run according to the scheduler’s algorithm.

When the server is restarted using the pbs_server -t hot command, jobs that were checkpointed and aborted upon shutdown are immediately rerun, before the scheduler selects which jobs to run.

9.3.7.5  Preemption Using Checkpoint

When a job is preempted via checkpointing, MoM runs the checkpoint_abort script, and PBS kills and requeues the job. When the scheduler elects to run the job again, the scheduler runs the restart script to restart the job from where it was checkpointed. For a description of using preemption, see section 4.8.33, “Using Preemption”, on page 228.
9.3.7.6  Holding a Job

When anyone uses the `qhold` command to hold a checkpointable job, MoM runs the checkpoint_abort script, which kills all job processes, and PBS requeues, and holds the job. A job with a hold on it must have the hold released via the `qrls` command in order to be eligible to run.

The following is the sequence of events when a job is held:

- MoM runs the checkpoint_abort script
- The job’s execution is halted
- The resources assigned to the job are released
- The job is placed in the Held state in the execution queue
- The job’s `Hold_Types` attribute is set appropriately

A held job is waiting in its queue. The following is the sequence of events when a held job is restarted:

- The hold is released by means of the `qrls` command; the job is now in the Queued state
- The job continues to wait in its queue until the scheduler schedules it for execution
- The scheduler selects the job for execution
- The job is sent to its original MoM for execution
- The MoM runs the restart script

9.3.7.6.1  Restrictions on Holding a Job

A job in the process of provisioning cannot be held.

The `qhold` command can be used on job arrays, but not on subjobs or ranges of subjobs.

If the job cannot be checkpointed and aborted, `qhold` simply sets the job’s `Hold_Types` attribute. The job continues to execute.

The checkpoint-abort script must terminate all job processes, or the `qhold` command will appear to hang.

9.3.7.7  Periodic Application Checkpoint

The snapshot checkpoint script can trigger checkpoint by a job’s application, if the application is written to support checkpointing itself. Note that an application may be designed to be checkpointed at specific stages in its execution, rather than at specific points in time. If an application can be usefully checkpointed at specific points in time, then snapshot checkpointing may be useful. See section 9.3.7.1, “Periodic Job Checkpointing”, on page 728.
Chapter 9  Making Your Site More Robust

9.3.7.8  Manual Application Checkpoint

When an application is checkpointed manually, the user triggers checkpointing by the application by sending the application a specific signal, or by creating a file.

9.3.8  Advice and Caveats

Multi-vnode jobs may cause network sockets to be opened between submission and execution hosts, and open sockets may cause a checkpointing script or tool to fail. The following use sockets:

- An interactive job, i.e. a job submitted using qsub -I, opens unprivileged sockets. qsub binds a socket to a port, then waits to accept a connection from MoM on that socket. Data from standard in is written to the socket and data from the socket is written to standard out.
- The pbs_demux process collects stdio streams from all tasks
- The pbsdsh program spawns tasks. The -o option to this command prevents it from waiting for spawned tasks to finish, so that no socket is left open to the MoM to receive task manager events. When the -o option is used, the shell must use some other method to wait for the tasks to finish. See “pbsdsh” on page 99 of the PBS Professional Reference Guide.

9.3.9  Accounting

If a job is checkpointed and requeued, the exit status passed to the epilogue and recorded in the accounting record is the following:

-12, meaning that the job was checkpointed and aborted

A checkpoint (“C”) record is written in the accounting log when the job is checkpointed and requeued, as when the qhold command is used, or the job is checkpointed and aborted.
9.4 Preventing Communication and Timing Problems

9.4.1 Introduction

PBS communicates with remote execution hosts in order to track their availability and manage the jobs running on them. PBS is dependent upon your network for this communication. If there are network outages, or if the execution node becomes too busy for MoM to be able to respond to the server’s queries, PBS will not be able to function properly. You can configure PBS to be better able to withstand these types of communication issues.

The following attributes and parameters control how PBS handles communication timing:

Table 9-9: Attributes and Parameters For Communication Timing

<table>
<thead>
<tr>
<th>Attribute or Parameter</th>
<th>Description</th>
<th>Cross Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server Attributes</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>job_requeue_timeout</td>
<td>Controls how long the process of requeueing a job is allowed to take</td>
<td>See section 9.4.3, “Setting Job Requeue Timeout”, on page 737</td>
</tr>
<tr>
<td>node_fail_requeue</td>
<td>Controls how long the server waits before requeueing or deleting a job when it loses contact with the MoM on the job’s primary execution host</td>
<td>See section 9.4.2, “Node Fail Requeue: Jobs on Failed Vnodes”, on page 734</td>
</tr>
<tr>
<td>rpp_retry</td>
<td>Maximum number of times the RPP library will try to re-send a packet before giving up.</td>
<td>See section 9.4.4, “Inter-daemon Communication”, on page 737</td>
</tr>
<tr>
<td>rpp_highwater</td>
<td>Maximum number of RPP packets allowed in transit at any one time.</td>
<td>See section 9.4.4, “Inter-daemon Communication”, on page 737</td>
</tr>
<tr>
<td><strong>MoM Configuration Parameters</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$max_load</td>
<td>Vnode is considered to be busy if it is above this load.</td>
<td>See section 9.4.5, “Managing Load Levels on Vnodes”, on page 738</td>
</tr>
</tbody>
</table>
9.4.2 Node Fail Requeue: Jobs on Failed Vnodes

The `node_fail_requeue` server attribute controls how long the server waits before requeueing or deleting a job when it loses contact with the MoM on the job’s primary execution host.

9.4.2.1 How Node Fail Requeue Works

You can specify how long the server waits after it loses contact with primary execution before deleting or requeueing her jobs. This behavior is controlled by the server’s `node_fail_requeue` attribute.

This attribute’s value is the delay between the time the server determines that the primary execution host cannot be contacted and the time it requeues the job, and does not include the time it takes to determine that the host is out of contact.

If this attribute is set to a value other than zero, and the server loses contact with an execution host, all jobs for which this is the primary execution host are requeued or deleted at the same time.

If `node_fail_requeue` is unset, and the host where primary execution is running fails, the server assumes that the job is still running until one of the following happens:

- The primary execution host comes back up and tells the server to requeue the job
- The job is manually rerun

Table 9-9: Attributes and Parameters For Communication Timing

<table>
<thead>
<tr>
<th>Attribute or Parameter</th>
<th>Description</th>
<th>Cross Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ideal_load</td>
<td>Vnode is considered to be not busy if it is below this load.</td>
<td>See section 9.4.5, “Managing Load Levels on Vnodes”, on page 738</td>
</tr>
<tr>
<td>$prologalarm</td>
<td>Maximum number of seconds the prologue and epilogue may run before timing out</td>
<td>See section 9.4.6, “Prologue &amp; Epilogue Running Time”, on page 741</td>
</tr>
<tr>
<td>Queue Attributes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>route_retry_time</td>
<td>Interval between retries at routing a job</td>
<td>See section 9.4.7, “Time Between Routing Retries”, on page 742</td>
</tr>
</tbody>
</table>
9.4.2.2 Effect Of Requeueing On Jobs

When a job is thus requeued, it retains its original place in its execution queue with its former priority. The job is usually the next job to be considered during scheduling, unless the relative priorities of the jobs in the queue have changed. This can happen when the job sorting formula assigns higher priority to another job, another higher-priority job is submitted after the requeued job started, this job’s owner has gone over their fairshare limit, etc.

Any resources that were being used by a job are freed when the job is requeued.

9.4.2.3 The node_fail_requeue Server Attribute

Format: Integer

9.4.2.3.i Allowable Values

The node_fail_requeue attribute can take these values:

Greater than zero
The server waits for the specified number of seconds after losing contact with a primary execution host, then attempts to contact the primary execution host, and if it cannot, requeues any jobs that can be rerun and deletes any jobs that cannot be rerun.

Zero
Jobs are not requeued; they are left in the Running state until the execution vnode is recovered, whether or not the server has contact with their Mother Superior.

Less than zero
The attribute is treated as if it were set to 1, and jobs are deleted or requeued after the server has been out of contact with Mother Superior for 1 second.

9.4.2.3.ii Default Value

The default value for this attribute is 310, meaning that when the server loses contact with an execution host, it waits for 310 seconds after losing contact with Mother Superior before requeueing or deleting jobs.

9.4.2.4 Where node_fail_requeue Applies

The server’s node_fail_requeue attribute applies only in the case where the server loses contact with Mother Superior.

When Mother Superior loses contact with a sister MoM, the job is immediately deleted or requeued.
9.4.2.5 Jobs Eligible to be Requeued

Jobs are eligible to be requeued if they meet either of the following criteria:

- The job's Rerunnable attribute is set to y
- The job did not begin execution, for example:
  - a multi-host job did not start on one or more vnodes
  - provisioning failed for the job

Jobs are ineligible to be requeued if their Rerunnable attribute is set to n and they have started execution.

See “Job Attributes” on page 375 of the PBS Professional Reference Guide and “Server Attributes” on page 316 of the PBS Professional Reference Guide.

9.4.2.6 Using node_fail_requeue

The number of seconds selected should be long enough to exceed any transient non-vnode failures, but short enough to requeue the job in a timely fashion. Transient non-vnode failures can prevent MoM from reporting back to the server before the server marks the vnode down.

These include:

- Network outages
- Vnode is too busy to respond, perhaps due to heavy swapping

Using this feature requires that you take the following into account:

- If the host where Mother Superior is running fails, and node_fail_requeue is unset, the server assumes that the job is still running until one of the following happens:
  - Mother Superior comes back up and tells the server to requeue the job
  - The job is manually rerun

If your site has hosts that fail and are not monitored, failed jobs may go unnoticed for a long time.

- If your network has temporary failures, and node_fail_requeue is set to a duration shorter than the outage, jobs will be unnecessarily requeued. This can be especially annoying when the job has been running for days.

9.4.2.7 Advice and Caveats

- If your site experiences frequent network failures or your execution hosts are often too busy to respond to the server, it is recommended that you either set node_fail_requeue to a value greater than the time MoM is unavailable, or set it to zero. This way jobs won’t
be requeued just because the network had a temporary outage or the vnode was too busy. Choose a value greater than both the longest likely network outage time and the time MoM is unavailable. For example, one site has set the value to 10 minutes, and another has set it to 15 minutes (900 seconds) to avoid problems due to swapping.

- The value shown in the server log for the time between losing communication and requeueing a job is sometimes one or two seconds less than the specified value.
- If the server is restarted when node_fail_requeue is set to a given value, node_fail_requeue retains that value. If the server is started when node_fail_requeue is unset, node_fail_requeue reverts to its default value.

9.4.3 Setting Job Requeue Timeout

When jobs are preempted via requeueing, the requeue can fail if the job being preempted takes longer than the allowed timeout. The time for requeueing includes post-processing such as staging files out, deleting files, and changing the job’s state from R to Q. See section 4.8.33, “Using Preemption”, on page 228. The time allowed for a job to be requeued is controlled by the job_requeue_timeout server attribute.

You can use qmgr to set the job_requeue_timeout server attribute to a value that works for the jobs at your site. This attribute is of type Duration, with a minimum allowed value of 1 second and a maximum allowed value of 3 hours. The default timeout is 45 seconds. See “Server Attributes” on page 316 of the PBS Professional Reference Guide.

9.4.4 Inter-daemon Communication

The Reliable Packet Protocol (RPP) is a library of functions used by PBS for reliable, flow-controlled, two-way data transmission. RPP is built on top of UDP. For more about RPP, see “RPP Library” on page 79 of the PBS Professional Programmer’s Guide.

9.4.4.1 Factors Affecting RPP Performance

Usually, RPP performance is affected at sites with many vnodes. In this case, the network is overwhelmed with all the UDP traffic, and packets are dropped.
9.4.4.2 When to Adjust RPP Settings

It can help to adjust rpp_retry when there are too many retries or failures showing up in log messages from any daemon. The log messages have this form:

/var/spool/PBS/server_logs/20100107:01/07/2010
21:41:39;0080;Server@surly;Node;rpp;total
(pkts=154,retries=120,fails=12) last 3601 secs
(pkts=154,retries=120,fails=12)

9.4.4.3 Finding Useful Values for RPP Settings

It is recommended that you start by setting rpp_retry to 30, and raise the value if that is insufficient. It is not recommended to set the value of rpp_retry to more than 100.

9.4.4.4 Server RPP Attributes

rpp_retry
The maximum number of times the RPP network library will try to send a UDP packet again before giving up. The number of retries is added to the original try, so if rpp_retry is set to 2, the total number of tries will be 3.

Format: Integer
Valid values: Greater than or equal to zero
Default: 10
Python type: int

rpp_highwater
The maximum number of RPP packets that can be in transit at any time.

Format: Integer
Valid values: Greater than or equal to one
Default: 64
Python type: int

9.4.5 Managing Load Levels on Vnodes

An overloaded execution host may end up too busy for MoM to respond to the server’s queries, and causing the server to mark the MoM as down.

PBS can track the state of each execution host, running new jobs on the host according to whether the host is marked busy or not.
This behavior is somewhat different from load balancing, described in section 4.8.27, “Using Load Balancing”, on page 193. In load balancing, the scheduler estimates how much load a job would produce, and will not place a job where doing so would put the load above the limit. When managing load levels on vnodes as described here, the scheduler uses the state of the vnode to determine whether to place a job on that vnode.

The state of the vnode is set by MoM, according to its load. You can set two load levels using the $max_load and $ideal_load MoM configuration parameters. When the load goes above $max_load, the vnode is marked as busy. When the load drops below $ideal_load, the vnode is marked free.

PBS does not run new jobs on vnodes under the following conditions:

- Vnodes that are marked busy
- Vnodes whose resources, such as ncpus, are already fully allocated
- Vnodes where the load is above $max_load, when load balancing is turned on. See section 4.8.27, “Using Load Balancing”, on page 193.
- Vnodes where running the job would cause the load to go above $max_load, when load balancing is turned on. See section 4.8.27, “Using Load Balancing”, on page 193.

The load used by MoM is the following:

- On UNIX/Linux, it is the raw one-minute averaged “loadave” returned by the operating system
- On Windows, it is based on the processor queue length

The $max_load and $ideal_load MoM configuration parameters are also used for cycle harvesting (see section 4.8.9.6, “Cycle Harvesting Based on Load Average”, on page 147) and load balancing (see section 4.8.27, “Using Load Balancing”, on page 193.)

MoM checks the load average on her host every 10 seconds.

When a vnode’s state changes, for example from free to busy, MoM informs the server.

### 9.4.5.1 Techniques for Managing Load

Whether or not you set $max_load, PBS will not run jobs requesting a total of more than the available number of CPUs, which is set in resources_available.ncpus. So for example if resources_available.ncpus is set to 4, and a job running on the vnode has requested 2 CPUs, PBS will not run jobs requesting a total of more than 2 CPUs.

#### 9.4.5.1.i Types of Workload

How you manage load depends on your workload. Some jobs do not lend themselves to sharing CPUs, but some jobs can share CPUs without being hindered. Most MPI jobs would be hindered if some processes had to wait because others were slowed by sharing a CPU. If you
need a job to have reproducible timing, it cannot share a CPU. Certain single-vnode jobs that alternate between CPU usage and I/O can share a CPU without being slowed significantly, thereby increasing throughput.

9.4.5.1.ii How Not To Share CPUs

For vnodes primarily running jobs that would be slowed or invalidated by sharing a CPU, have PBS assign jobs according to the number of available CPUs, so that there is no sharing of CPUs. Set \texttt{resources\_available.ncpus} to the number of available CPUs. Do not set $\texttt{max\_load}$ or $\texttt{ideal\_load}$.

9.4.5.1.iii How To Share CPUs

For vnodes running only jobs that can share CPUs, you can have PBS manage jobs according to the load on the vnodes, not the number of CPUs. This is called oversubscribing the CPUs. Set \texttt{resources\_available.ncpus} to a value greater than the actual number of CPUs, such as two or three times the actual number. Set $\texttt{max\_load}$ to a reasonable value so that PBS will run new jobs until $\texttt{max\_load}$ is reached. Set $\texttt{ideal\_load}$ to the minimum load that you want on the vnode.

9.4.5.1.iv Suspending Jobs on Overloaded Vnodes

You can specify that MoM should suspend jobs when the load goes above $\texttt{max\_load}$, by adding the \texttt{suspend} argument to the $\texttt{max\_load}$ parameter. See section \texttt{“$max\_load <load> [suspend]”}, on page 741. In this case, MoM suspends all jobs on the vnode until the load drops below $\texttt{ideal\_load}$, then resumes them. This option is useful only when the source of the load includes work other than PBS jobs. This option is not recommended when the load is due solely to PBS jobs, because it can lead to the vnode cycling back and forth between being overloaded, being marked busy, suspending all jobs, being marked free, then starting all jobs, being overloaded, and so on.

9.4.5.2 Caveats and Recommendations

- It is recommended that the value for $\texttt{ideal\_load}$ be lower than the value for $\texttt{max\_load}$. The value for $\texttt{ideal\_load}$ should be low enough that new jobs are not run before existing jobs are done using the vnode’s spare load.

- If you set only one of $\texttt{max\_load}$ and $\texttt{ideal\_load}$, for example you set $\texttt{max\_load}$, but not $\texttt{ideal\_load}$, PBS sets the other to the same value.

- Do not allow reservations on hosts where $\texttt{max\_load}$ and $\texttt{ideal\_load}$ are configured. Set the \texttt{resv\_enable} vnode attribute on these hosts to \texttt{False}.

- If you are using cycle harvesting via load balancing, be careful with the settings for $\texttt{ideal\_load}$ and $\texttt{max\_load}$. You want to make sure that when the workstation owner is using the machine, the load on the machine triggers MoM to report being busy, and that...
PBS does not start any new jobs while the user is working. See section 4.8.9.6, “Cycle Harvesting Based on Load Average”, on page 147.

### 9.4.5.2.i Allowing Non-job Processes on Execution Host

If you wish to run non-PBS processes on a host, you can prevent PBS from using more than you want on that host. Set the $ideal_load and $max_load MoM configuration parameters to values that are low enough to allow other processes to use some of the host.

### 9.4.5.3 Load Configuration Parameters

#### $ideal_load <load>
MoM parameter. Defines the load below which the vnode is not considered to be busy. Used with the $max_load parameter.

Example:

```
$ideal_load 1.8
```

Format: Float
No default

#### $max_load <load> [suspend]
MoM parameter. Defines the load above which the vnode is considered to be busy. Used with the $ideal_load parameter.

If the optional `suspend` argument is specified, PBS suspends jobs running on the vnode when the load average exceeds $max_load, regardless of the source of the load (PBS and/or logged-in users).

Example:

```
$max_load 3.5
```

Format: Float
Default: number of CPUs

### 9.4.6 Prologue & Epilogue Running Time

Each time the scheduler runs a job, it waits for the prologue to finish before it runs another job. In order to prevent a hung prologue from halting job execution, prologues and epilogues are only allowed to run for a specified amount of time before PBS kills them. The running time is specified in the $prologalarm MoM configuration parameter. The default value for this parameter is 30 seconds.
9.4.6.1 Prologue Timeout Configuration Parameter

$prologalarm <timeout>

Defines the maximum number of seconds the prologue and epilogue may run before timing out.

Example:

$prologalarm 30

Format: Integer

Default: 30

9.4.7 Time Between Routing Retries

If the network is flaky, PBS may not be able to route a job from a routing queue to the destination queue. If all destination queues for a routing queue are at capacity, a job in a routing queue remains where it is. The time between routing retries is controlled by the route_retry_time queue attribute.

If the network experiences long outages, you may wish to set the time between retries to a sufficiently long time that PBS is not wasting cycles attempting to route jobs.

If jobs in a routing queue are not being routed because the destination queues are full, and most jobs are long-running jobs, you may wish to set the time between retries so that attempts are infrequent. It is recommended that the time between retries be no longer than the longest time acceptable to have an open slot in an execution queue.

9.4.7.1 Routing Retry Attribute

route_retry_time

Time delay between routing retries. Typically used when the network between servers is down. Used only with routing queues.

Format: Integer seconds

Default: 30 seconds

Python type: pbs.duration
9.5 Reservation Fault Tolerance

If the vnodes associated with an advance reservation or the soonest occurrence of a standing reservation become unavailable, PBS marks the advance or standing reservation as *degraded*. PBS attempts to reconfirm degraded reservations by finding replacements for vnodes that have become unavailable. Usable vnodes from the original reservation are retained.

States of available vnodes:

- *free*
- *busy*
- *job-exclusive*
- *job-sharing*
- *job-busy*

States of unavailable vnodes:

- *down*
- *offline*
- *provisioning*
- *stale*
- *state-unknown, down*
- *unresolvable*
- *wait-provisioning*

The reservation’s state becomes *RESV_DEGRADED*, abbreviated DG, and its substate becomes *RESV_DEGRADED*.

If vnodes associated with an occurrence later than the soonest occurrence of a standing reservation become unavailable, the reservation stays in state *RESV_CONFIRMED*, but its substate becomes *RESV_DEGRADED*.

During the time that a degraded advance reservation or the soonest occurrence of a degraded standing reservation is running, its state is *RESV_RUNNING*, and its substate is *RESV_DEGRADED*.

For a table of degraded reservation states and substates, see “Degraded Reservation Substates” on page 416 of the PBS Professional Reference Guide. For a table of numeric values for reservation states and substates, see “Reservation States” on page 415 of the PBS Professional Reference Guide.
9.5.1 Reconfirming Reservations

PBS attempts to reconfirm degraded reservations by finding replacements for vnodes that have become unavailable. Usable vnodes from the original reservation are retained. PBS starts periodically trying to reconfirm the reservation after the reservation becomes degraded, and stops trying before the reservation’s start time. For standing reservations, PBS tries to reconfirm subsequent occurrences if the soonest occurrence could not be reconfirmed.

PBS periodically tries to reconfirm a degraded reservation until either:

- The reservation is confirmed
- The cutoff time is reached

A degraded reservation has a read-only reservation attribute called \texttt{reserve_retry}, whose value is the next time at which the reservation is due to be reconfirmed.

9.5.1.1 Attributes Affecting Reservation Reconfirmation

\texttt{reserve_retry_cutoff}

Server attribute. The time period before the reservation start time during which PBS does not attempt to reconfirm a degraded reservation. When this value is changed, all degraded reservations use the new value.

Cannot be read by user or Operator.

Format: Integer (seconds)

Values: Must be greater than \texttt{zero}

Default: 7200 (2 hours)

Python attribute value type: int

\texttt{reserve_retry_init}

Server attribute. The amount of time after a reservation becomes degraded that PBS waits before attempting to reconfirm the reservation. When this value is changed, only reservations that become degraded after the change use the new value.

Cannot be read by user or Operator.

Format: Integer seconds

Values: Must be greater than \texttt{zero}

Default: 7200 (2 hours)

Python attribute value type: int
9.5.1.2 Configuring Reservation Reconfirmation

You can configure the amount of time between when a reservation becomes degraded and when PBS first tries to reconfirm it, using the server attribute \texttt{reserve_retry_init}. If you unset this attribute, it reverts to its default value. If you change this attribute, previously degraded reservations are unaffected; only reservations that become degraded after the change have the new attribute value applied to them.

PBS stops trying to reconfirm a reservation for a configurable amount of time before that reservation’s start time. The amount of time between when PBS stops trying to reconfirm a reservation and that reservation’s start time is specified in the server attribute \texttt{reserve_retry_cutoff}. If you unset this attribute, it reverts to its default value. If you change this attribute, all degraded reservations use the new value.

9.5.1.3 Reconfirming Standing Reservations

PBS attempts to reconfirm each soonest occurrence of a standing reservation. This means that if the first occurrence of a standing reservation becomes degraded, and PBS is unable to reconfirm it, then after its reservation period has passed, PBS will try to reconfirm the second occurrence of the standing reservation, and so on. PBS will try to reconfirm each occurrence of the standing reservation until either the reservation is confirmed or there are no more occurrences.

9.5.2 Allocating New Vnodes

Once new vnodes are allocated for a reservation:

\begin{itemize}
\item The reservation has been confirmed
\item The state and substate of the reservation are \texttt{RESV\_CONFIRMED}
\item The reservation’s \texttt{resv\_nodes} attribute lists the new vnodes
\end{itemize}

9.5.3 Restarting the Server

When the server is restarted, reservations are assumed confirmed until associated vnodes are recognized as unavailable. If any reservations become degraded after a server restart, PBS sets the time when the reservation becomes degraded to the time of the restart. If a vnode is set \texttt{offline} before the restart, it is considered unavailable after the restart, so all its associated reservations become degraded.
Chapter 9  Making Your Site More Robust

9.5.4  Logging Degraded Reservation Information

The server logs a message when a vnode associated with a confirmed reservation becomes unavailable, at event class 0x0100:

An attempt to reconfirm reservation will be made on <ctime timestamp>

The server logs a message for each attempt at reconfirming a reservation, at event class 0x0100:

Next attempt to reconfirm reservation will be made on <ctime timestamp>

The scheduler logs degraded reservations at event class 0x0200:

Reservation is in degraded mode, <x> out of <y> vnodes are unavailable;
<list of vnodes down>

9.6  Preventing File System Problems

9.6.1  Avoid Filling Location of Temp Files for PBS Components

When the location used by PBS components to store temporary files becomes full, various failures may result, including jobs not initializing properly. To help avoid this, you can set the root directory for these files to a location less likely to fill up. See section 12.10, “Temporary File Location for PBS Components”, on page 890.

In addition, we recommend periodic cleaning of this location.

9.6.2  Avoid Filling Filesystem with Log Files

You must avoid having log files fill up the available space. You may have to rotate and archive log files frequently to ensure that adequate space remains available. See “Adequate Space for Logfiles” on page 6 in the PBS Professional Installation & Upgrade Guide.
9.7 Built-in Robustness

9.7.1 OOM Killer Protection

If the system hosting a PBS daemon or data service runs low on memory, the system may use an out-of-memory killer (OOM killer) to terminate processes. The PBS daemons and data service are protected from being terminated by an OOM killer.
Chapter 9  Making Your Site More Robust
Integrations

This chapter covers integrating PBS with MPIs and other tools, as well as with platforms requiring special attention. This chapter is intended for the PBS administrator.

10.1 Integration with MPI

PBS Professional is integrated with several implementations of MPI. When PBS is integrated with an MPI, PBS can track resource usage, control jobs, clean up job processes, and perform accounting for all of the tasks run under the MPI.

When PBS is not integrated with an MPI, PBS can track resource usage, clean up processes, and perform accounting only for processes running on the primary vnode. This means that accounting and tracking of CPU time and memory aren't accurate, and job processes on sister vnodes cannot be signaled.

10.1.1 Prerequisites

Before you integrate an MPI with PBS, the MPI must be working by itself. For example, you must make sure that all required environment variables are set correctly for the MPI to function.

10.1.2 Types of Integration

PBS provides support for integration for many MPIs. You can integrate MPIs with PBS using the following methods:

- Wrapping the MPI with a PBS-supplied script which uses the TM (task manager) inter-
face to manage job processes. PBS supplies a master script to wrap any of several MPIs. See section 10.1.6, “Integration by Wrapping”, on page 754

- PBS supplies wrapper scripts for some MPIs, for wrapping those MPIs by hand. See section 10.1.9, “Integration By Hand”, on page 759
- For non-integrated MPIs, job scripts can integrate the MPIs on the fly using the `pbs_tmrsh` command. Note that a PBS job script that uses `mpirun` with `pbs_tmrsh` cannot be used outside of PBS. See section 10.1.5, “Integration on the Fly using the pbs_tmrsh Command”, on page 752 and "Integrating an MPI on the Fly using the pbs_tmrsh Command", on page 115 of the PBS Professional User’s Guide.
- Some MPIs can be compiled to use the TM interface. See section 10.1.4, “Integration Using the TM Interface”, on page 752
- Some MPIs require users to call `pbs_attach`, for example SGI MPT. See section 10.1.9.4, “Integrating SGI MPT”, on page 766.

To integrate an MPI with PBS, you use just one of the methods above. The method you choose depends on the MPI. The following table lists the supported MPIs, how to integrate them, and gives links to the steps involved and any special notes about that MPI:

**Table 10-1: List of Supported MPIs**

<table>
<thead>
<tr>
<th>MPI Name</th>
<th>Versions</th>
<th>Method</th>
<th>Integration Steps</th>
<th>MPI-specific Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP MPI</td>
<td>1.08.03 2.0.0</td>
<td>Use <code>pbs_mpihp</code></td>
<td>&quot;Steps to Integrate HP MPI or Platform MPI&quot;</td>
<td>&quot;Integrating HP MPI and Platform MPI&quot;</td>
</tr>
<tr>
<td>IBM POE</td>
<td>AIX 5.x, 6.x</td>
<td>Use <code>pbsrun_wrap</code></td>
<td>&quot;Wrapping an MPI Using the pbsrun_wrap Script&quot;</td>
<td>&quot;Integration by Wrapping&quot;</td>
</tr>
<tr>
<td>Intel MPI</td>
<td>2.022 3 4</td>
<td>Use <code>pbsrun_wrap</code></td>
<td>&quot;Wrapping an MPI Using the pbsrun_wrap Script&quot;</td>
<td>&quot;Integration by Wrapping&quot;</td>
</tr>
<tr>
<td>LAM MPI</td>
<td>6.5.9 Deprecated.</td>
<td>Use <code>pbs_mpiilm</code> and <code>pbs_lamboot</code></td>
<td>&quot;Wrapping LAM MPI 6.5.9&quot;</td>
<td>&quot;Integrating LAM MPI and Open MPI&quot;</td>
</tr>
</tbody>
</table>
## Integrations

Chapter 10

### Table 10-1: List of Supported MPIs

<table>
<thead>
<tr>
<th>MPI Name</th>
<th>Versions</th>
<th>Method</th>
<th>Integration Steps</th>
<th>MPI-specific Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAM MPI</td>
<td>7.0.6 7.1.1</td>
<td>Compile with TM</td>
<td>&quot;Integration Using the TM Interface&quot;</td>
<td>&quot;Integrating LAM MPI and Open MPI&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPICH-P4</td>
<td>1.2.5 1.2.6 1.2.7</td>
<td>Use pbs_mpirun</td>
<td>&quot;Steps to Integrate MPICH-P4&quot;</td>
<td>&quot;Integrating MPICH-P4&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPICH-GM (MPICH 1.2.6.14b)</td>
<td>Use pbsrun_wrap</td>
<td></td>
<td>&quot;Wrapping an MPI Using the pbsrun_wrap Script&quot;</td>
<td>&quot;Integration by Wrapping&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPICH-MX</td>
<td>Use pbsrun_wrap</td>
<td></td>
<td>&quot;Wrapping an MPI Using the pbsrun_wrap Script&quot;</td>
<td>&quot;Integration by Wrapping&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPICH2</td>
<td>1.0.3 1.0.5 1.0.7</td>
<td>Use pbsrun_wrap</td>
<td>&quot;Wrapping an MPI Using the pbsrun_wrap Script&quot;</td>
<td>&quot;Integration by Wrapping&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVAPICH, MVAPICH2</td>
<td>1.2, 1.8</td>
<td>Use pbsrun_wrap</td>
<td>&quot;Wrapping an MPI Using the pbsrun_wrap Script&quot;</td>
<td>&quot;Integration by Wrapping&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open MPI</td>
<td>1.4.x</td>
<td>Compile with TM</td>
<td>&quot;Integration Using the TM Interface&quot;</td>
<td>&quot;Integrating LAM MPI and Open MPI&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Platform MPI</td>
<td>8.0</td>
<td>Use pbs_mpihp</td>
<td>&quot;Steps to Integrate HP MPI or Platform MPI&quot;</td>
<td>&quot;Integrating HP MPI and Platform MPI&quot;</td>
</tr>
</tbody>
</table>
10.1.3 Transparency to the User

Many MPIs can be integrated with PBS in a way that is transparent to the job submitter. This means that a job submitter can use the same MPI command line inside and outside of PBS. All of the MPIs listed above can be made to be transparent.

10.1.4 Integration Using the TM Interface

PBS provides an API to the PBS task manager, or TM, interface. You can configure an MPI to use the PBS TM interface directly.

When a job process is started on a sister vnode using the TM interface, the sister vnode’s MoM starts the process and the primary vnode’s MoM has access to job process information.

Two MPIs that we know can be compiled with the TM interface are LAM MPI and Open MPI.

10.1.5 Integration on the Fly using the pbs_tmrsh Command

If using a non-integrated MPI, job submitters can integrate an MPI on the fly by using the pbs_tmrsh command. This command emulates rsh, but uses the TM interface to talk directly to pbs_mom on sister vnodes. The pbs_tmrsh command informs the primary and sister MoMs about job processes on sister vnodes. PBS can track resource usage for all job processes.

<table>
<thead>
<tr>
<th>MPI Name</th>
<th>Versions</th>
<th>Method</th>
<th>Integration Steps</th>
<th>MPI-specific Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGI MPT</td>
<td>Any</td>
<td>Optional: Use mpiexec, or users put pbs_attach in mpirun command line</td>
<td>&quot;Steps to Integrate SGI MPT&quot;</td>
<td>&quot;Integrating SGI MPT&quot;</td>
</tr>
</tbody>
</table>
Job submitters use this command by setting the appropriate environment variable to \texttt{pbs_tmrsh}. For example, to integrate MPICH, set \texttt{P4_RSHCOMMAND} to \texttt{pbs_tmrsh}. For details, see "Integrating an MPI on the Fly using the \texttt{pbs_tmrsh} Command", on page 115 of the PBS Professional User’s Guide.

The following figure illustrates how the \texttt{pbs_tmrsh} command can be used to integrate an MPI on the fly:

\begin{figure}
\centering
\includegraphics[width=\textwidth]{integrations.png}
\caption{PBS knows about processes on vnodes 2 and 3, because \texttt{pbs_tmrsh} talks directly to \texttt{pbs_mom}, and \texttt{pbs_mom} starts the processes on vnodes 2 and 3.}
\end{figure}

\section*{10.1.5.1 Caveats for the \texttt{pbs_tmrsh} Command}

- This command cannot be used outside of a PBS job; if used outside a PBS job, this command will fail.
- The \texttt{pbs_tmrsh} command does not perform exactly like \texttt{rsh}. For example, you cannot pipe output from \texttt{pbs_tmrsh}; this will fail.
10.1.6 Integration by Wrapping

Wrapping an MPI means replacing its `mpirun` or `mpiexec` with a script which calls the original executable and, indirectly, `pbs_attach`. Job processes are started by `rsh` or `ssh`, but the `pbs_attach` command informs the primary and sister MoMs about the processes, so that PBS has control of the job processes. See “`pbs_attach`” on page 45 of the PBS Professional Reference Guide.

PBS provides a master script called `pbsrun_wrap` that you use to wrap many MPIs. PBS supplies special wrapper scripts so that you can wrap other MPIs by hand.

The following figure shows how a wrapped `mpirun` call works:

![Diagram of `mpirun` call with wrapping](image)

Figure 10-2: The job script calls the link that has the name of the original `mpirun`
10.1.6.1 Wrap the Correct Instance

When you wrap an MPI, make sure that you are wrapping the first instance of the name found in the user's search path. This is the one returned by the 'which' command on Linux.

For example, on IBM systems poe is installed as follows:

```
-rwxrwxrwx 1 root system 31 Apr 14 19:21 /usr/bin/poe -> /usr/lpp/ppe.poe/bin/poe
```

And 'which' returns the following:

```
bash-2.05b# which poe
/usr/bin/poe
```

Here, you must wrap the link, not the binary.

10.1.7 Wrapping an MPI Using the pbsrun_wrap Script

The master script is the pbsrun_wrap command, which takes two arguments: the mpirun to be wrapped, and a PBS-supplied wrapper. The pbsrun_wrap command neatly wraps the original mpirun so that everything is transparent for the job submitter. See “pbsrun_wrap” on page 123 of the PBS Professional Reference Guide, and “pbsrun” on page 106 of the PBS Professional Reference Guide.

The pbsrun_wrap command does the following:

- Renames the original, named mpirun.<flavor>, to mpirun.<flavor>.actual
- Instantiates the wrapper as pbsrun.<flavor>
- Creates a link named mpirun.<flavor> that calls pbsrun.<flavor>
- Creates a link so that pbsrun.<flavor> calls mpirun.<flavor>.actual

10.1.7.1 Passing Arguments

Any mpirun version/flavor that can be wrapped has an initialization script ending in ".init", found in $PBS_EXEC/lib/MPI:

```
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init
```

When executed inside a PBS job, the pbsrun.<flavor> script calls a version-specific initialization script which sets variables to control how the pbsrun.<flavor> script uses options passed to it. For example, pbsrun.<flavor> calls $PBS_EXEC/lib/MPI/pbsrun.<flavor>.init to manage the arguments passed to it. You can modify the .init scripts to specify which arguments should be retained, ignored, or transformed.
When the mpirun wrapper script is run inside a PBS job, then it translates any mpirun call of the form:

`mpirun [options] <executable> [args]`

into

`mpirun [options] pbs_attach [special_option_to_pbs_attach] <executable> [args]`

where [special options] refers to any option needed by pbs_attach to do its job (e.g. -j $PBS_JOBID).

See “Options” on page 107 of the PBS Professional Reference Guide for a description of how to customize the initialization scripts.

### 10.1.7.2 Restricting MPI Use to PBS Jobs

You can specify that a wrapped MPI can be used only inside of PBS, by using the `-s` option to the `pbsrun_wrap` command. This sets the `strict_pbs` option in the initialization script (e.g. `pbsrun.ch_gm.init`, etc...) to 1 from the default of 0. This means that the mpirun being wrapped by pbsrun will be executed only when it is called inside a PBS environment. Otherwise, the user gets the following error:

```
Not running under PBS
exiting since strict_pbs is enabled; execute only in PBS
```

By default, when the wrapper script is executed outside of PBS, a warning is issued about "not running under PBS", but it proceeds as if the actual program had been called in standalone fashion.

### 10.1.7.3 Format of pbsrun_wrap Command

The `pbsrun_wrap` command has this format:

`pbsrun_wrap [-s] <path_to_actual_mpirun> pbsrun.<keyword>`

Make sure that you wrap the correct instance of the mpirun. If a user’s job script would call a link, wrap the link. See section 10.1.6.1, “Wrap the Correct Instance”, on page 755.

### 10.1.7.4 Actions During Wrapping

The `pbsrun_wrap` script instantiates the pbsrun wrapper script as `pbsrun.<mpirun version/flavor>` in the same directory where pbsrun is located, and sets up the link to the actual mpirun call via the symbolic link:

```
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link
```
For example, running:

```
pbsrun_wrap /opt/mpich-gm/bin/mpirun.ch_gm pbsrun.ch_gm
```

causes the following actions:

- Save original mpirun.ch_gm script:
  ```
  mv /opt/mpich-gm/bin/mpirun.ch_gm /opt/mpich-gm/bin/mpirun.ch_gm.actual
  ```

- Instantiate pbsrun wrapper script as pbsrun.ch_gm:
  ```
  cp $PBS_EXEC/bin/pbsrun $PBS_EXEC/bin/pbsrun.ch_gm
  ```

- Link mpirun.ch_gm to actually call pbsrun.ch_gm:
  ```
  ln -s $PBS_EXEC/bin/pbsrun.ch_gm /opt/mpich-gm/bin/mpirun.ch_gm
  ```

- Create a link so that pbsrun.ch_gm calls mpirun.ch_gm.actual:
  ```
  ln -s /opt/mpich-gm/bin/mpirun.ch_gm.actual $PBS_EXEC/lib/MPI/pbsrun.ch_gm.link
  ```

### 10.1.7.5 Requirements

The mpirun being wrapped must be installed and working on all the vnodes in the PBS cluster.

### 10.1.7.6 Caveats and Restrictions

- For MPIs that are wrapped using pbsrun_wrap, the maximum number of ranks that can be launched in a job is the number of entries in the $PBS_NODEFILE.

- MVAPICH2 must use the “mpd” process manager if it is to be integrated with PBS. During the configuration step when you build MVAPICH2, set the “process manager” setting to `mpd`, as follows:
  ```
  --with-pm=mpd
  ```

Other process managers such as "hydra" and "gforker" may not work correctly with PBS.
## 10.1.7.7 Links to Wrapper Script Information

The following table lists the links to the description of each wrapper script used by `pbsrun_wrap`:

<table>
<thead>
<tr>
<th>MPI Wrapper</th>
<th>Link to Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM POE</td>
<td>“IBM’s poe: pbsrun.poe” on page 120 of the PBS Professional Reference Guide.</td>
</tr>
<tr>
<td>MPICH2</td>
<td>See “MPICH2’s mpirun: pbsrun.mpich2” on page 116 of the PBS Professional Reference Guide.</td>
</tr>
<tr>
<td>MPICH-GM with MPD</td>
<td>“MPICH-GM’s mpirun (mpirun.mpd) with MPD: pbsrun.gm_mpd” on page 113 of the PBS Professional Reference Guide.</td>
</tr>
<tr>
<td>MPICH-MX with MPD</td>
<td>“MPICH-MX’s mpirun (mpirun.mpd) with MPD: pbsrun.mx_mpd” on page 115 of the PBS Professional Reference Guide.</td>
</tr>
<tr>
<td>MPICH-MX with rsh/ssh</td>
<td>“MPICH-MX’s mpirun (mpirun.ch_mx) with rsh/ssh: pbsrun.ch_mx” on page 113 of the PBS Professional Reference Guide.</td>
</tr>
<tr>
<td>MVAPICH</td>
<td>“MVAPICH1’s mpirun: pbsrun.mvapich1” on page 118 of the PBS Professional Reference Guide.</td>
</tr>
<tr>
<td>MVAPICH2</td>
<td>“MVAPICH2’s mpiexec: pbsrun.mvapich2” on page 119 of the PBS Professional Reference Guide.</td>
</tr>
<tr>
<td>Intel MPI</td>
<td>“Intel MPI’s mpirun: pbsrun.intelmpi” on page 117 of the PBS Professional Reference Guide.</td>
</tr>
</tbody>
</table>
10.1.7.8 Wrapping Multiple MPIs with the Same Name

You may want more than one MPI environment with the same name, for example a 32-bit and a 64-bit version of MPICH2.

1. Create two new MPICH2 initialization scripts by copying that for MPICH2:
   
   ```
   # cd $PBS_EXEC/lib/MPI
   # cp pbsrun.mpich2.init.in pbsrun.mpich2_32.init.in
   # cp pbsrun.mpich2.init.in pbsrun.mpich2_64.init.in
   ```

2. Then wrap them:
   
   ```
   # pbsrun_wrap <path to 32-bit MPICH2>/bin/mpirun pbsrun.mpich2_32
   # pbsrun_wrap <path to 64-bit MPICH2>/bin/mpirun pbsrun.mpich2_64
   ```

   Calls to `<path to 32-bit MPICH2>/bin/mpirun` will invoke `/usr/pbs/bin/pbsrun.mpich2_32`. The 64-bit version is invoked with calls to `<path to 64-bit MPICH2>/bin/mpirun`.

10.1.7.9 See Also


10.1.8 Unwrapping MPIs Using the pbsrun_unwrap Script

You can also use the matching `pbsrun_unwrap` command to unwrap the MPIs you wrapped using `pbsrun_wrap`.

For example, you can unwrap the two MPICH2 MPIs from 10.1.7.8 above:

```
# pbsrun_unwrap pbsrun.mpich2_32
# pbsrun_unwrap pbsrun.mpich2_64
```

See “pbsrun_unwrap” on page 122 of the PBS Professional Reference Guide.

10.1.9 Integration By Hand

For MPIs that must be wrapped by hand, PBS supplies wrapper scripts which call the original and use `pbs_attach` to give MoM control of jobs.
Wrapping an MPI by hand yields the same result as wrapping using `pbsrun_wrap`, but you must perform the steps by hand.

Wrapping by hand involves the following steps (which are the same steps taken by `pbsrun_wrap`):

- You rename the original MPI command
- You create a link whose name is the same as the original MPI command; this link calls the wrapper script
- You edit the wrapper script to call the original MPI command
- You make sure that the link to the wrapper script(s) is available to each user’s PATH.

The following table lists MPIs, their wrapper scripts, and a link to instructions:

<table>
<thead>
<tr>
<th>MPI Name</th>
<th>Script Name</th>
<th>Link to Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP MPI</td>
<td>pbs_mpihp</td>
<td>section 10.1.9.1, “Integrating HP MPI and Platform MPI”, on page 760</td>
</tr>
<tr>
<td>LAM MPI 6.5.9</td>
<td>pbs_mpilam</td>
<td>section 10.1.9.2, “Integrating LAM MPI and Open MPI”, on page 762</td>
</tr>
<tr>
<td>LAM MPI 6.5.9</td>
<td></td>
<td>Deprecated.</td>
</tr>
<tr>
<td>MPICH</td>
<td>pbs_mpirun</td>
<td>section 10.1.9.3, “Integrating MPICH-P4”, on page 764</td>
</tr>
<tr>
<td>Platform MPI</td>
<td>pbs_mpihp</td>
<td>section 10.1.9.1, “Integrating HP MPI and Platform MPI”, on page 760</td>
</tr>
<tr>
<td>SGI MPT</td>
<td>mpiexec</td>
<td>section 10.1.9.4, “Integrating SGI MPT”, on page 766</td>
</tr>
</tbody>
</table>

### 10.1.9.1 Integrating HP MPI and Platform MPI

PBS supplies a wrapper script for HP MPI and Platform MPI called `pbs_mpihp`. The `pbs_mpihp` script allows PBS to clean up job processes, track and limit job resource usage, and perform accounting for all job processes.

You can make `pbs_mpihp` transparent to users; see the instructions that follow.
10.1.9.1.i  Steps to Integrate HP MPI or Platform MPI

Make sure that you wrap the correct instance of the MPI. If a user’s job script would call a link, wrap the link. See section 10.1.6.1, “Wrap the Correct Instance”, on page 755.

The pbs_mpirun command looks for a link with the name PBS_EXEC/etc/pbs_mpihp that points to the HP mpirun. The pbs_mpihp command follows this link to HP’s mpirun. Therefore, the wrapping instructions are different from the usual. See “pbs_mpihp” on page 64 of the PBS Professional Reference Guide for more information on pbs_mpihp.

1. Rename HP’s mpirun:
   ```bash
cd <MPI installation location>/bin
mv mpirun mpirun.hp
```

2. Link the user-callable mpirun to pbs_mpihp:
   ```bash
cd <MPI installation location>/bin
ln -s $PBS_EXEC/bin/pbs_mpihp mpirun
```

3. Create a link to mpirun.hp from PBS_EXEC/etc/pbs_mpihp. pbs_mpihp will call the real HP mpirun:
   ```bash
cd $PBS_EXEC/etc
ln -s <MPI installation location>/bin/mpirun.hp pbs_mpihp
```

10.1.9.1.ii Setting Up rsh and ssh Commands

When wrapping HP MPI with pbs_mpihp, note that rsh is the default used to start the mpids. If you wish to use ssh or something else, be sure to set the following or its equivalent in $PBS_HOME/pbs_environment:

```bash
PBS_RSHCOMMAND=ssh
```

10.1.9.1.iii Restrictions and Caveats for HP MPI and Platform MPI

- The pbs_mpihp script can be used only on HP-UX and Linux.
- The HP mpirun or mpiexec must be in the job submitter’s PATH.
- The version of the HP mpirun or mpiexec must be HPMPI or Platform.
- Under the wrapped HP MPI, the job’s working directory is changed to the user’s home directory.
Chapter 10  Integrations

10.1.9.2  Integrating LAM MPI and Open MPI

The 7.x LAM MPI and all Open MPI versions allow you to compile the MPI with the PBS TM interface. We recommend compiling 7.x LAM MPI and all Open MPI versions with the TM module. You can either compile the later LAM with TM or wrap it, but not both. (You can wrap the newer versions if you want, but compiling yields better results.)

All versions of LAM MPI and Open MPI can be transparent to the job submitter.

The 6.5.9 version of LAM MPI requires wrapping for integration. Support for the 6.5.9 version of LAM MPI is deprecated.

10.1.9.2.i  Compiling LAM MPI 7.x/Open MPI with the TM Module

To integrate 7.x LAM MPI with PBS, compile it with the --with-boot-tm=/usr/pbs option. Next, check laminfo to confirm that the the SSI line that says tm is there.

If the TM interface library is in the standard location, PBS_EXEC/lib/, Open MPI will find it and use it. You need to explicitly configure with TM only if it’s in a non-standard location.

To integrate Open MPI with PBS, configure Open MPI with the --with-tm command-line option to the configure script. For example:

    ./configure --prefix=/opt/openmpi/1.4.4 --with-tm=${PBS_EXEC}

        make
    make install

After you compile LAM MPI or Open MPI on one host, make it available on every execution host that will use it, by means of shared file systems or local copies.

For the Open MPI website information on compiling with the TM option, see:

http://www.open-mpi.org/faq/?category=building#build-rte-tm

10.1.9.2.ii  Wrapping LAM MPI 6.5.9

Support for the 6.5.9 version of LAM MPI is deprecated.

PBS provides wrapper scripts so that you can integrate LAM MPI 6.5.9 with PBS by hand. The pbs_mpilam script is used in place of mpirun, and the pbs_lamboot script replaces lamboot. The pbs_lamboot and pbs_mpilam scripts allow PBS to clean up job processes, track and limit job resource usage, and perform accounting for all job processes. You make LAM calls transparent to the user by allowing them to use unchanged lamboot and lamhalt calls in their scripts.

The PBS command pbs_lamboot replaces the standard lamboot command in a PBS LAM MPI job, for starting LAM software on each of the PBS execution hosts. Usage is the same as for LAM lamboot. All arguments except for bhost are passed directly to lamboot. PBS will issue a warning saying that the bhost argument is ignored by PBS since
input is taken automatically from $PBS_NODEFILE. The pbs_lamboot command can be
instructed to boot the hosts using the tm module by setting the LAM_MPI_SSI_boot environment variable to tm, or by passing an argument to pbs_lamboot that contains “-ssi boot tm”. In this case, the pbs_lamboot program does not redundantly consult the $PBS_NODEFILE.

The PBS command pbs_mpilam replaces the standard mpirun command in a PBS LAM
MPI job, for executing programs. It attaches the user’s processes to the PBS job. This allows
PBS to collect accounting information, and to manage the processes. Usage is the same as for
LAM mpirun. All options are passed directly to mpirun. If the where argument is not
specified, pbs_mpilam will try to run the user’s program on all available CPUs using the C
keyword.

Make sure that you wrap the correct instance of the MPI. If a user’s job script would call a
link, wrap the link. See section 10.1.6.1, “Wrap the Correct Instance”, on page 755.

• You wrap LAM lamboot using pbs_lamboot.
  1. Install LAM MPI into /usr/local/lam-6.5.9.
  2. Rename LAM lamboot to lamboot.lam:
      mv /usr/local/lam-6.5.9/bin/lamboot /usr/local/lam-6.5.9/bin/lamboot.lam
  3. Edit pbs_lamboot to change “lamboot” call to “lamboot.lam”:
  4. Create a link for pbs_lamboot named lamboot:
      cd /usr/local/lam-6.5.9/bin
      ln -s PBS_EXEC/bin/pbs_lamboot lamboot
      At this point, using “lamboot” will actually invoke pbs_lamboot.
• You wrap LAM mpirun using the pbs_mpilam script.
  1. Install LAM MPI into /usr/local/lam-6.5.9.
  2. Rename LAM mpirun to mpirun.lam:
      mv /usr/local/lam-6.5.9/bin/mpirun /usr/local/lam-6.5.9/bin/mpirun.lam
  3. Edit pbs_mpilam to change “mpirun” call to “mpirun.lam”
  4. Create a link for pbs_mpilam named mpirun:
      cd /usr/local/lam-6.5.9/bin
      ln -s PBS_EXEC/bin/pbs_mpilam mpirun

For more information on pbs_lamboot and pbs_mpilam, see “pbs_lamboot” on page 54
Chapter 10  Integrations

10.1.9.2.iii Setting up rsh and ssh Commands
If you intend to use ssh, you should set either LAMRSH or LAM_SSI_rsh_agent to the value "ssh -x", except under SuSE Linux, where it should be "ssh -n".

10.1.9.2.iv Setting up Environment Variables
Set the LAM_MPI_SSI_boot environment variable to tm so that pbs_lamboot boots the hosts from the tm module.

10.1.9.2.v Verifying Use of TM Interface
To see whether your Open MPI installation has been configured to use the TM interface:

```bash
% ompi_info | grep tm
MCA ras: tm (MCA v2.0, API v2.0, Component v1.3)
MCA plm: tm (MCA v2.0, API v2.0, Component v1.3)
```

10.1.9.2.vi See Also
See www.lam-mpi.org for more information about LAM MPI.
See http://www.open-mpi.org/faq/?category=building#build-rte-tm for information about building Open MPI with the TM option.

10.1.9.3 Integrating MPICH-P4
PBS supplies a wrapper script called pbs_mpirun for integrating MPICH-P4 with PBS by hand. The pbs_mpirun script allows PBS to clean up job processes, track and limit job resource usage, and perform accounting for all job processes.
You can make pbs_mpirun transparent to job submitters. See the following steps.

10.1.9.3.i Restrictions
• The pbs_mpirun command can be used only with MPICH using P4 on Linux.
• User names must be identical across vnodes.

10.1.9.3.ii Options for pbs_mpirun
The usage for pbs_mpirun is the same as mpirun except for the listed options. All other options are passed directly to mpirun:

```
-machinefile
```

The value for this option is generated by pbs_mpirun. The value used for the -machinefile option is a temporary file created from the PBS_NODEFILE in the format expected by mpirun.
Integrations

If the -machinefile option is specified on the command line, a warning is output saying "Warning, -machinefile value replaced by PBS".

-np

The default value for the -np option is the number of entries in PBS_NODEFILE.

10.1.9.3.iii  Steps to Integrate MPICH-P4

To make pbs_mpirun transparent to the user, replace standard mpirun with pbs_mpirun. Make sure that you wrap the correct instance of the MPI. If a user’s job script would call a link, wrap the link. See section 10.1.6.1, “Wrap the Correct Instance”, on page 755.

- Install MPICH-P4 into <path to mpirun>
- Rename mpirun to mpirun.std:
  
  mv <path to mpirun>/mpirun <path to mpirun>/mpirun.std

- Create link called mpirun in <path to mpirun> that points to pbs_mpirun
  
  ln -s <path to pbs_mpirun>/pbs_mpirun mpirun

- Edit pbs_mpirun to change the call to mpirun so that it calls mpirun.std

At this point, using mpirun actually invokes pbs_mpirun.

10.1.9.3.iv  Setting Up Environment Variables and Paths

- For pbs_mpirun to function correctly for users who require the use of ssh instead of rsh, you can do one of the following:
  
  - Set PBS_RSHCOMMAND in the login environment
  - Set P4_RSHCOMMAND externally to the login environment, then have job submitters pass the value to PBS via qsub(1)'s -v or -V arguments:

    
    qsub -v P4_RSHCOMMAND=ssh ...
    or

    qsub -V ...

  
  - Set P4_RSHCOMMAND in the pbs_environment file in PBS_HOME and then advise users to not set P4_RSHCOMMAND in the login environment

- Make sure that PATH on remote machines contains PBS_EXEC/bin. Remote machines must all have pbs_attach in the PATH.

- The PBS_RSHCOMMAND environment variable should not be set by the user.

- When using SuSE Linux, use “ssh -n” in place of “ssh”.

PBS Professional 12.2 Administrator’s Guide
10.1.9.4 Integrating SGI MPT

PBS supplies its own mpiexec on the Altix running supported versions of ProPack or SGI Performance Suite, in order to provide a standard interface for use by job submitters. This mpiexec calls the standard SGI mpirun. If users call this mpiexec, PBS will manage, track, and cleanly terminate multi-host MPI jobs.

If job submitters call SGI MPT directly, they must use pbs_attach in their job scripts in order to give PBS the same control over jobs; see the SGI documentation.

MPI jobs can be launched across multiple Altixes. PBS users can run MPI jobs within specific partitions.

When job submitters use mpiexec in their job scripts, SGI MPT is transparent. Jobs run normally whether the PBS-supplied mpiexec is called inside or outside of PBS.

10.1.9.4.i Supported Platforms

The PBS-supplied mpiexec runs on the Altix running supported versions of ProPack or SGI Performance Suite.

10.1.9.4.ii Steps to Integrate SGI MPT

Make sure that the PBS-supplied mpiexec is in each user’s PATH.

10.1.9.4.iii Invoking SGI MPT

PBS uses the MPI-2 industry standard mpiexec interface to launch MPI jobs within PBS. If executed on a non-Altix system, PBS’s mpiexec will assume it was invoked by mistake. In this case it will use the value of PATH (outside of PBS) or PBS_O_PATH (inside PBS) to search for the correct mpiexec and if one is found, exec it.

The name of the array to use when invoking mpirun is user-specifiable via the PBS_MPI_SGIARRAY environment variable.

10.1.9.4.iv Using SGI MPT Over InfiniBand

To use InfiniBand, set the MPI_USE_IB environment variable to 1.

10.1.9.4.v Using CSA with SGI MPT

PBS support for CSA on SGI systems is no longer available. The CSA functionality for SGI systems has been removed from PBS.

10.1.9.4.vi Prerequisites

- In order to run single-host or multi-host jobs, the SGI Array Services must be correctly configured. An Array Services daemon (arrayd) must run on each host that will run MPI processes. For a single-host environment, arrayd only needs to be installed and
• Integrations

Chapter 10

activated. However, for a multi-host environment where applications will run across hosts, the hosts must be properly configured to be an array.

• Altix systems communicating via SGI's Array Services must all use the same version of the sgi-mpt and sgi-arraysvcs packages. Altix systems communicating via SGI’s Array Services must have been configured to interoperate with each other using the default array. See SGI’s array_services(5) man page.

• “rpm -qi sgi-arraysvcs” should report the same value for Version on all systems.

• “rpm -qi sgi-mpt” should report the same value for Version on all systems.

• “chkconfig array” must return “on” for all systems

• /usr/lib/array/arrayd.conf must contain an array definition that includes all systems.

• /usr/lib/array/arrayd.auth must be configured to allow remote access:

  The “AUTHENTICATION NOREMOTE” directive must be commented out or removed.

  Either “AUTHENTICATION NONE” should be enabled or keys should be added to enable the SIMPLE authentication method.

• If any changes have been made to the arrayd configuration files (arrayd.auth or arrayd.conf), the array service must be restarted.

• rsh(1) must work between the systems.

• PBS uses SGI's mpirun(1) command to launch MPI jobs. SGI's mpirun must be in the standard location.

• The location of pbs_attach(8B) on each vnode of a multi-vnode MPI job must be the same as it is on the mother superior vnode.

10.1.9.4.vii Environment Variables

• The PBS mpiexec script sets the PBS_CPUSET_DEDICATED environment variable to assert exclusive use of the resources in the assigned cpuset.

• If the PBS_MPI_DEBUG environment variable's value has a nonzero length, PBS will write debugging information to standard output.

• If the PBS_MPI_SGIARRAY environment variable is present, the PBS mpiexec will use its value as the name of the array to use when invoking mpirun.

• The PBS_ENVIRONMENT environment variable is used to determine whether mpiexec is being called from within a PBS job.

• If it was invoked by mistake, the PBS mpiexec uses the value of PBS_O_PATH to search for the correct mpiexec.

• To use InfiniBand, set the MPI_USE_IB environment variable to 1.
10.1.10 How Processes are Started Using MPI and PBS

10.1.10.1 Starting Processes under Non-integrated MPIs

The following figure illustrates how processes are started on sister vnodes when using a non-integrated MPI:

Figure 10-3: PBS does not know about processes on vnodes 2 and 3, because those processes were generated outside of the scope of PBS.
10.1.10.2 Starting Processes under Wrapped MPIs

The following figure illustrates how processes are started on sister vnodes when using a wrapped MPI:

![Diagram showing process flow for starting processes under wrapped MPIs]

Figure 10-4: PBS knows about processes on vnodes 2 and 3, because `pbs_attach` tells those MoMs which processes belong to which jobs
10.1.10.3 Starting Processes Under MPIs Employing the TM Interface

The following figure illustrates how processes are started on sister vnodes when using an MPI that employs the TM interface:

Figure 10-5: PBS knows about processes on vnodes 2 and 3, because the TM interface talks directly to pbs_mom, and pbs_mom starts the processes on vnodes 2 and 3

10.1.11 Limit Enforcement with MPI

PBS can enforce the following for a job using MPI:

- Per-process limits via `setrlimit(2)` on sister vnodes
  - The `setrlimit` process limit can be enforced only when using an MPI that
employs the TM interface directly, which is LAM and Open MPI only

- Limits set via MoM parameters, e.g. `cpuburst` and `cpuaverage`, on sister vnodes
  - PBS can enforce these limits using any integrated MPI
- Job-wide limits such as `cput`, `mem`
  - PBS can enforce job-wide limits using any integrated MPI

Once a process is started, process limits cannot be changed.

### 10.1.12 Restrictions and Caveats for MPI Integration

- There are no MPIs that run on Windows that are integrated with PBS, meaning that resource tracking and accounting under Windows happens only for processes on the primary vnode.
- The `pbs_attach` command is not supported under Windows.
- Be sure to wrap the correct instance of the MPI. See section 10.1.6.1, “Wrap the Correct Instance”, on page 755
- Some applications write scratch files to a temporary location in `tmppdir`. The location of `tmppdir` is host-dependent. If you are using an MPI that is not integrated with the PBS TM interface, and your application needs scratch space, the location of `tmppdir` for the job should be consistent across execution hosts. You can specify the root for `tmppdir` in the MoM’s `$tmpdir` configuration parameter. PBS sets the job’s `TMPDIR` environment variable to the temporary directory it creates for the job.

### 10.2 Support for IBM AIX

You can wrap IBM `poe` with a PBS-supplied wrapper script so that PBS can track resource usage, signal jobs, and perform accounting for all job processes run under `poe`.

PBS automatically uses the HPS if it is available. PBS can run jobs using InfiniBand. PBS jobs can use User Space Mode on the HPS or on Infiniband switches. PBS can track the resources for MPI, LAPI programs or a mix of MPI and LAPI programs. LoadLeveler is not required in order to use InfiniBand switches in User Space mode.

PBS supports large page mode on AIX. See section 10.2.8, “Support for Large Page Mode on AIX”, on page 776.
10.2.1 Overview of Integrating IBM POE with PBS

To integrate POE with PBS, take the following steps:

- Wrap IBM’s `poe` with a PBS-supplied script. This script uses the PBS TM interface to give PBS control of job resources and processes, so that PBS can track resources, signal jobs, and perform accounting for all job processes. You use the `pbsrun_wrap` command to integrate PBS with IBM `poe`; see section 10.1.6, “Integration by Wrapping”, on page 754.

- If you want all InfiniBand jobs to have the number of required windows calculated by the PBS server (recommended), enable this as described in section 10.2.2.1.i, “Requirements for Server Calculation”, on page 773.

- Enable scheduling of jobs using network windows, to ensure that the number of jobs on the InfiniBand switch is correct. See section 10.2.6, “Enabling Scheduling of Jobs Using Network Windows”, on page 776.

- If the default for the number of switch network IDs for each chunk is unacceptable, you can set a value. See section 10.2.4, “Setting Value for Number of Switch Network IDs”, on page 775.

- To limit the number of instances that can be requested for each chunk, see section 10.2.5, “Setting Maximum Value for Number of Instances”, on page 775.

- If an HPS is available, configure a resource to represent it. We recommend calling this resource `hps`. See section 10.2.7, “Allowing Users to Request HPS”, on page 776.

10.2.2 Managing Network Windows

PBS has two methods for computing the number of network windows to be used by a job. The server can track all network windows, or the MoM can perform a calculation for a default value. We recommend having the server do the calculation, because the server can avoid oversubscribing the windows. Oversubscription can lead to job rejection.

PBS has a built-in host-level read-only resource called `netwins` for representing the network windows on a high performance switch. This resource can be used to schedule jobs that use the InfiniBand switch.

Each chunk in a job needs to have a value for the `netwins` resource.

By default, the `netwins` resource is included in the “resources:” line in `PBS_HOME/sched_priv/sched_config`.

If the server calculates the value for `netwins`, the value calculated is logged in the server log at the event class of `pbs.LOG_WARNING`. 
10.2.2.1 Calculation of Windows by Server

10.2.2.1.i Requirements for Server Calculation

For the server to calculate the value for netwins, the PBS_GET_IBWINS environment variable must be set to 1 in the job’s environment. You can do this for all jobs, or the job submitter can set this variable for one job or multiple jobs. We recommend that you do this for all jobs.

- To set the environment variable for all jobs (recommended), do one of the following:
  - Include “-v PBS_GET_IBWINS = 1” in the server’s default_qsub_arguments attribute:

    ```
    set server default_qsub_arguments += “-v PBS_GET_IBWINS = 1”
    ```
  - Use a submission hook to add “-v PBS_GET_IBWINS = 1” to the resource request of each job that will use windows

- To set the environment variable for multiple jobs, the job submitter sets PBS_GET_IBWINS = 1 in their shell environment, and uses the -V option to the qsub command.
  - csh:

    ```
    setenv PBS_GET_IBWINS 1
    ```
  - bash:

    ```
    PBS_GET_IBWINS = 1
    export PBS_GET_IBWINS
    ```

- To set the environment variable for one job, the job submitter uses the “-v PBS_GET_IBWINS = 1” option to the qsub command.

10.2.2.1.ii Formula Used by Server for Windows Calculation

The PBS server calculates the value of netwins to be assigned to each chunk as follows:

\[
\text{netwins} = \frac{\text{number of mpiprocs in chunk} \times \text{number of switch networks} \times \text{number of instances} \times \text{msg_api multiplier}}{\text{number of mpiprocs in chunk}}
\]

- The value for the mpiprocs resource

\[
\text{number of switch networks}
\]

- Defaults to 2
- The value of the custom server-level resource called PBS_ibm_ib_default_networks. If this resource is defined, this value takes precedence over the default value.
Chapter 10  Integrations

number of instances

- Defaults to 1
- Value passed via `qsub` option `-v MP_INSTANCES`, or value passed via `qsub` option `-v` when `MP_INSTANCES` is set in user’s shell environment. If the value of `MP_INSTANCES` is given, it takes precedence over the default value.

msg_api multiplier

- Defaults to 1
- Value is 2 if the user specifies “mpi,lapl” either via the `-v MP_MSG_API` option to `qsub`, or via the `-v` option to `qsub` when `MP_MSG_API` is set in the user’s shell environment. If the user specifies “mpi, lapl”, this takes precedence over the default value.

10.2.2.2 Using Default Number of Network Windows

If the PBS server has not assigned a value for `netwins` for a job that will use the InfiniBand switch, the PBS MoM calculates the number of network windows to configure on each switch as follows:

\[
\text{number of mpiproc} \times \text{number of switch networks}
\]

MoM queries any high performance network that is attached to her execution host.

10.2.2.2.i Caveats for Default Number of Windows

If the user requests more than one MPI process per CPU, this default number of network windows may be too large, and prevent use of the InfiniBand switch.

10.2.3 Caveats for InfiniBand Network

We recommend that you do not set a value for `netwins`, via either `qmgr` or MoM configuration files.
10.2.4 Setting Value for Number of Switch Network IDs

The number of switch network IDs per chunk defaults to 2. If the default is not acceptable:

1. Define a custom server-level resource called PBS_ibm_ib_default_networks in the resourcedef file:
   
   PBS_ibm_ib_default_networks type=long flag=r

2. Restart the server; see section 5.14.3.1, “Restarting the Server”, on page 332.

3. Set the new resource to the number of switch network IDs that all the InfiniBand adapters are connected with:
   
   qmgr -c "s s resources_available.PBS_ibm_ib_default_networks = <new value>"

10.2.5 Setting Maximum Value for Number of Instances

The number of instances per chunk defaults to 1. Users can request a different number of instances by setting the MP_INSTANCES environment variable to that number. To set the maximum number of instances per chunk that users can request:

1. Define a custom server-level resource called PBS_ibm_ib_max_instances. Put the following in the resourcedef file:
   
   PBS_ibm_ib_max_instances type=long flag=r

2. Restart the server; see section 5.14.3.1, “Restarting the Server”, on page 332.

3. Calculate the number of instances to be allowed per chunk. A conservative number can be calculated by:
   
   PBS_ibm_ib_max_instances = ((number of adapters per node) * (number of network windows per adapter)) / ((number of CPUs per node) * 2)

4. Set this resource to the maximum number of instances per process:
   
   qmgr -c "s s resources_available.PBS_ibm_ib_max_instances = <max number windows>"

If the user requests more than the maximum specified in PBS_ibm_ib_max_instances, the requested value is replaced with the maximum value.
10.2.6 Enabling Scheduling of Jobs Using Network Windows

PBS can schedule jobs using the netwins resource, ensuring that the load on the InfiniBand switch is correct. To schedule jobs using this resource:

- The environment variable PBS_GET_IBWINS must be set to 1 in the job’s environment, as in section 10.2.2.1.i, “Requirements for Server Calculation”, on page 773.
- The netwins resource must be in the resources line in PBS_HOME/sched/priv/sched_config.

10.2.7 Allowing Users to Request HPS

To allow users to request the HPS, you need to set up a custom resource for the HPS. Users can then request this resource in their select statements. We recommend that this resource is called hps.

The kind of resource you use depends on how you want to schedule jobs on the machine with the HPS. If you don’t need to explicitly limit the number of jobs using the HPS, use a host-level Boolean resource, and set it to True only on the vnodes that use the HPS. We recommend this method. See section 5.14.2, “Defining New Custom Resources”, on page 323.

If you do need to explicitly limit the number of jobs using the HPS, use a static host-level consumable resource. Set this to the number of chunks or jobs that should use the HPS at one time. It is more difficult to make sure that jobs request the correct value for the HPS resource. It is relatively easy for a job that requests “place=pack” to also request one additional hps resource, however, if a job requests chunks that are “place=scatter”, it becomes more difficult to count the resource correctly. See section 5.14.5.2, “Static Host-level Resources”, on page 339.

If you have some machines in the complex that are not on the HPS, be sure that those machines have their hps resource set to False or 0.

```bash
# qmgr -c 'set node not_ibm resources_available.hps = false'
```

10.2.8 Support for Large Page Mode on AIX

PBS Professional supports Large Page Mode on AIX. You do not need to do anything to enable this behavior. Certain applications (like many FEA Solvers) can benefit from using large page support. This allows programs to do considerably less page “thrashing”.

776
Setting the PBS environment to request large page mode is not recommended because every process started by a job will use large page mode. It is better for the user to explicitly request large page mode for the processes that should use large page mode.

10.2.9 Errors and Logging

The PBS MoM may encounter the following errors, and write these error messages to the MoM logfile:

• Error return from `nrt_status_adapter`
  The message shows the adapter name, error code from `nrt_status_adapter`, and errno result:
  
  01/24/2007 18:57:26;0008;pbs_mom;Job;1.cyan;nrt_status_adapter(iba0) err 4; I/O error

• Error return from `nrt_load_table`
  The message shows the adapter name, error code from `nrt_load_table`, and errno result:
  
  01/24/2007 18:57:26;0008;pbs_mom;Job;1.cyan;nrt_load_table(iba0) err 1; Invalid input parameter

• Error return from `nrt_unload_window`
  The message shows the adapter name, error code from `nrt_unload_window`, and errno result:
  
  01/24/2007 18:57:26;0008;pbs_mom;Job;1.cyan;nrt_unload_window(iba0) err 12; I/O error

• Error return from `nrt_adapter_resources`
  The message shows the adapter name, error code from `nrt_adapter_resources`, and errno result:
  
  01/24/2007 18:57:26;0008;pbs_mom;Job;1.cyan;nrt_adapter_resources(iba0) err 3; I/O error

10.3 Support for Cray Systems

PBS provides features designed to support the Cray. This section of the documentation describes the special behavior of PBS on Cray systems only.
Chapter 10  Integrations

10.3.1  Introduction

PBS provides support for Cray systems by providing the following:

- PBS automatically defines vnodes for Cray compute nodes
- PBS automatically sets resources and attributes for vnodes representing Cray nodes
- PBS automatically creates custom resources that correspond to Cray resources
- Cray users can submit jobs through PBS using the PBS select and place statements.

10.3.2  Relationship of PBS Vnodes to Cray Nodes

PBS represents each login node as a vnode, and each NUMA node as a vnode. A compute node containing a single NUMA node is represented as a single vnode. The Cray inventory uses the term “segment” to refer to a NUMA node. A PBS MoM runs on each login node; this MoM manages the vnodes representing the compute nodes associated with the login node. On systems with multiple login nodes, each MoM on each login node manages every compute node. When this is the case, each compute node is reported by more than one login node. The `Mom` attribute of a vnode representing a compute node contains the hostname of each login node reporting the compute node. Each hostname is the FQDN or the short name of the reporting login node, depending on whatever is returned by the DNS.

10.3.2.1  How PBS Handles Changes in Cray Inventory

10.3.2.1.i  Reporting Changes in Vnode List

If a previously-reported vnode is no longer reported when the vnode list is created, because it is no longer available in the vnode definition file and the inventory, it is missing from the vnode list. The server marks missing vnodes as `stale`.

PBS provides a Boolean MoM configuration option that allows you to specify whether MoM tells the server that a vnode is missing. When the `$vnodedef_additive` MoM configuration option in `PBS_HOME/mom_priv/config` is `True`, MoM does not tell the server that any vnodes are missing. This means that the server does not mark missing vnodes as `stale`. When `$vnodedef_additive` is `False`, MoM tells the server that vnodes are missing, and the server marks the missing vnodes as `stale`. The default value for `$vnodedef_additive` for a MoM managing a Cray is `False`. The default value for other systems is `True`.

When a compute node goes down and the ALPS inventory no longer reports it, the vnode is marked `stale` once PBS queries for the inventory. This happens when PBS fails to confirm a Cray reservation. If you bring the compute node back up, you can HUP the MoM in order to make the vnode usable by PBS. Alternatively, if PBS fails to make a Cray reservation, MoM will re-read the inventory and re-create the vnode list.
Each compute node can be managed by more than one login node. This means that the vnodes representing the NUMA nodes in a compute node can have more than one MoM reporting them.

A vnode representing a NUMA node is marked *stale* when any of the MoMs that reported the vnode stop reporting it.

The state of a vnode representing a NUMA node that is managed by more than one login node is not changed by a MoM going down, unless all of its MoMs are down. A vnode representing a NUMA node is marked *down* when all of the MoMs that manage the vnode are *down*.

### 10.3.2.1.ii When MoMs Report Conflicting Information

When one or more MoMs report information about a vnode, and one or more MoMs do not list the vnode, PBS uses the information from the MoMs that still report the vnode.

When more than one MoM reports information about a vnode, and the information conflicts, PBS uses the most recent information.

### 10.3.2.1.iii Vnode Resources on Stale Vnodes

If the ALPS inventory no longer reports information for a vnode, and the vnode is not defined in a vnode definition file, all resource and attribute information for that vnode is removed or set to zero, and the vnode is marked *stale*.

If the ALPS inventory no longer reports information for a vnode, but the vnode is defined in a vnode definition file, the vnode’s attributes and resources retain their settings and the vnode is not marked *stale*.

### 10.3.3 Requirements

- PBS requires CLE 2.2 or higher.
- For a compute node to be managed by the PBS MoM, the node must be in batch mode, and in state *UP*.

### 10.3.4 Restrictions

- A Cray compute node cannot be used by more than one application at the same time.
- PBS does not report *cput* or *mem* for jobs running on a Cray compute node. Note that this prevents *cput* from being used for checkpointing.
10.3.5 Resources, Parameters, Etc.

PBS provides built-in and custom resources for the Cray. PBS also provides some built-in resources for all platforms that have specific uses on the Cray.

10.3.5.1 Built-in Resources for All Platforms

**accelerator**
Indicates whether this vnode is associated with an accelerator. Host-level. Can be requested only inside of a select statement. On Cray, this resource exists only when there is at least one associated accelerator. On Cray, this is set to True when there is at least one associated accelerator whose state is \textit{UP}. On Cray, set to False when all associated accelerators are in state \textit{DOWN}. Used for requesting accelerators.

Format: \textit{Boolean}
Python type: bool

**accelerator_memory**
Indicates amount of memory for accelerator(s) associated with this vnode. Host-level. Can be requested only inside of a select statement. On Cray, PBS sets this resource only on vnodes with at least one accelerator with state = \textit{UP}. For Cray, PBS sets this resource on the 0th NUMA node (the vnode with \textbf{PBScrayseg}=0), and the resource is shared by other vnodes on the compute node.

For example, on vnodeA\_2\_0:

```
resources\_available.accelerator\_memory=4196mb
```

On vnodeA\_2\_1:

```
resources\_available.accelerator\_memory=@vnodeA\_2\_0
```
Consumable.

Format: size
Python type: pbs.size

**accelerator\_model**
Indicates model of accelerator(s) associated with this vnode. Host-level. On Cray, PBS sets this resource only on vnodes with at least one accelerator with state = \textit{UP}. Can be requested only inside of a select statement. Non-consumable.

Format: String
Python type: str
naccelerators
Indicates number of accelerators on the host. Host-level. On Cray, should not be requested for jobs; PBS does not pass the request to ALPS. On Cray, PBS sets this resource only on vnodes whose hosts have at least one accelerator with state = $UP$. PBS sets this resource to the number of accelerators with state = $UP$. For Cray, PBS sets this resource on the 0th NUMA node (the vnode with PBScrayseg=0), and the resource is shared by other vnodes on the compute node.
For example, on vnodeA_2_0:

```
resources_available.naccelerators=1
```
On vnodeA_2_1:

```
resources_available.naccelerators=vnodeA_2_0
```
Can be requested only inside of a select statement, but should not be requested. Consumable.
Format: Long
Python type: int

nchunk
Used on all platforms. This is the number of chunks requested between plus symbols in a select statement. For example, if the select statement is `-lselect 4:ncpus=2+12:ncpus=8`, the value of nchunk for the first part is `4`, and for the second part it is `12`. The nchunk resource cannot be named in a select statement; it can only be specified as a number preceding the colon, as in the above example. When the number is omitted, nchunk is `1`.
Non-consumable.
This resource can be used to specify the default number of chunks at the server or queue (replacing mppwidth.)
Example:

```
Qmgr: set queue myqueue default_chunk.nchunk=2
```
Settable by Manager and Operator; readable by all.
This resource cannot be used in server and queue resources_min and resources_max.
Format: Integer
Python type: int
Default value: 1
10.3.5.2 Built-in Resources for the Cray

**vttype**

This resource represents the type of the vnode. Automatically set by PBS to one of two specific values for Cray vnodes. Has no meaning for non-Cray vnodes. Can be requested only inside of a select statement.

Non-consumable.

Format: *String array*

Automatically assigned values for Cray vnodes:

- **cray_compute**
  
  This vnode represents part of a compute node.

- **cray_login**
  
  This vnode represents a login node.

Default value: None

Python type: str

10.3.5.3 Custom Resources for the Cray

PBS creates custom resources that represent various Cray information. You do not need to do anything to create these resources.

**PBScrayhost**

On CLE 2.2, this is set to “default”.

On CLE 3.0 and higher, used to delineate a Cray system, containing ALPS, login nodes running PBS MoMs, and compute nodes, from a separate Cray system with a separate ALPS. Non-consumable. The value of **PBScrayhost** is set to the value of **mpp_host** for this system.

Format: *String*

Default: CLE 2.2: “default”; CLE 3.0 and higher: None

**PBScraylabel_<label name>**

Tracks labels applied to compute nodes. For each label on a compute node, PBS creates a custom resource whose name is a concatenation of **PBScraylabel_** and the name of the label. PBS sets the value of the resource to True on all vnodes representing the compute node.

Format: **PBScraylabel_<label name>**

For example, if the label name is Blue, the name of this resource is **PBScraylabel_Blue**.
Integrations

Chapter 10

Format: Boolean
Default: None

PBScraynid
Used to track the node ID of the associated compute node. All vnodes representing a particular compute node share a value for PBScraynid. Non-consumable.
The value of PBScraynid is set to the value of node_id for this compute node. Non-consumable.
Format: String
Default: None

PBScrayorder
Used to track the order in which compute nodes are listed in the Cray inventory. All vnodes associated with a particular compute node share a value for PBScrayorder. Non-consumable.
Vnodes for the first compute node listed are assigned a value of 1 for PBScrayorder. The vnodes for each subsequent compute node listed are assigned a value one greater than the previous value.
Format: Integer
Default: None

PBScrayseg
Tracks the segment ordinal of the associated NUMA node. For the first NUMA node of a compute host, the segment ordinal is 0, and the value of PBScrayseg for the associated vnode is 0. For the second NUMA node, the segment ordinal is 1, PBScrayseg is 1, and so on. Non-consumable.
Format: String
Default: None

10.3.5.4 Scheduler Attribute

do_not_span_psets
Specifies whether or not the scheduler requires the job to fit within one of the existing placement sets. When do_not_span_psets is set to True, the scheduler will require the job to fit within a single existing placement set. The scheduler checks all placement sets, whether or not they are currently in use. If the job fits in a currently-used placement set, the job must wait for the placement set to be available. If the job cannot fit within a single placement set, it will not run.
When this attribute is set to *False*, the scheduler will first attempt to place the job in a single placement set, but if it cannot, it will allow the job to span placement sets, running on whichever vnodes can satisfy the job’s resource request.

Format: *Boolean*

Default value: *False* (This matches behavior of PBS 10.4 and earlier)

Example: To require jobs to fit within one placement set:

```
Qmgr: set sched do_not_span_psets=True
```

### 10.3.5.5 MoM Configuration Options

**$alps_client**

MoM runs this command to get the ALPS inventory. Must be full path to command. Set during installation.

Format: path to command

Default: Unset

**$alps_release_timeout <timeout>**

Specifies the amount of time that PBS tries to release an ALPS request before giving up. After this amount of time has passed, PBS stops trying to release the ALPS reservation, the job exits, and the job’s resources are released. PBS sends a HUP to the MoM so that she rereads the ALPS inventory to get the current available ALPS resources.

We recommend that the value for this parameter be greater than the value for `suspectbegin`.

Format: Seconds, specified as positive integer.

Default: 600 (10 minutes)

**$vnodedef_additive**

Specifies whether MoM considers a vnode that appeared previously either in the inventory or in a vnode definition file, but that does not appear now, to be in her list of vnodes.

When `$vnodedef_additive` is *True*, MoM treats missing vnodes as if they are still present, and continues to report them as if they are present. This means that the server does not mark missing vnodes as *stale*.

When `$vnodedef_additive` is *False*, MoM does not list missing vnodes, the server’s information is brought up to date with the inventory and vnode definition files, and the server marks missing vnodes as *stale*.

Visible in configuration file on Cray only.

Format: *Boolean*
10.3.6 Automatic Configuration

10.3.6.1 Vnode List Creation

You must create a vnode for each login node, but PBS automatically creates the vnodes representing NUMA nodes. PBS creates a list of vnodes by reading the Cray inventory and any vnode definition files. PBS does this automatically when any of the following happens:

- Startup of the MoM
- The MoM is HUPed
- PBS queries the Cray for the inventory (e.g., when PBS fails to confirm a Cray reservation)
- PBS times out on trying to release an ALPS request.

10.3.6.2 Automatic Configuration of Compute Node Vnodes

PBS automatically creates one or more vnodes to represent each Cray compute node. Each vnode represents one Cray NUMA node.

10.3.6.3 Vnode Attribute and Resource Settings

PBS automatically sets the values of certain vnode attributes and resources. Vnode attribute and resource settings are derived from values returned in the inventory, according to the following rules:

- \texttt{resources\_available.accelerator}:
  - Set to \texttt{True} when this vnode’s host has at least one accelerator in state \texttt{UP}.

- \texttt{resources\_available.accelerator\_memory}:
  - Set to the value of \texttt{basil\_accelerator\_gpu.memory}.
  - Set on 0th NUMA node’s vnode; shared on other vnodes.

- \texttt{resources\_available.accelerator\_model}:
  - Set to the value of \texttt{basil\_accelerator\_gpu.family}.

- \texttt{resources\_available.naccelerators}:
  - Number of \texttt{Accelerator} entries for the host that are in state \texttt{UP}.
  - Set on 0th NUMA node’s vnode; shared on other vnodes.

- \texttt{resources\_available.host}:
  - Values of node’s \texttt{mpp\_host} and \texttt{node\_id} are concatenated.
Chapter 10  Integrations

Format: <mpp_host>_<node_id>

Example: Given a compute node where mpp_host = examplehost and node_id = 8, resources_available.host is set to examplehost_8.

On CLE 2.2, “default” replaces value of mpp_host.

resources_available.PBScrayhost
On CLE 2.2, set to “default”.
On CLE 3.0 and higher, set to value of mpp_host

resources_available.PBScraylabel_<label name>
For each label on a compute node, PBS creates a custom Boolean resource, and sets it to True on all vnodes representing that compute node. The format for the name of these resources is PBScraylabel_<label>. For example, if the label is Blue, then the name of the Boolean resource is PBScraylabel_Blue.

resources_available.PBScraynid
The value of PBScraynid is set to the value of node_id for this compute node.

resources_available.PBScrayorder
Set to the position in the Cray node list of the associated node. If this vnode’s associated node was nth in the node list, the value of PBScrayorder is n.

resources_available.PBScrayseg
Set to the segment ordinal of the associated NUMA node.

resources_available.vntype
On compute nodes, set to cray_compute
On internal login nodes, set to cray_login

Mom vnode attribute
This is the canonical hostname of the login node where MoM runs.

Name of vnode
Value of node’s mpp_host, node_id, and segment ordinal are concatenated.

Format: <mpp_host>_<node_id>_<segment>

Example: Given segment 0, node_id = 8 and mpp_host = examplehost, the vnode name is examplehost_8_0.
On CLE 2.2, “default” replaces value of mpp_host.

sharing vnode attribute
Set to force_exclhost
10.3.6.4 MoM Parameter Settings

$alps_release_timeout <timeout>
Specifies the amount of time that PBS tries to release an ALPS request before giving up. We recommend that the value for this parameter be greater than the value for suspectbegin.
Format: Seconds, specified as positive integer.
Default: 600 (10 minutes)

$vnode$edef_additive MoM configuration option
PBS automatically sets the value of the $vnode$edef_additive MoM configuration option to False on any MoM on a login node. See section 10.3.5.5, “MoM Configuration Options”, on page 784.

10.3.6.5 Scheduler Attribute Settings

do_not_span_psets
This attribute is set to False by default. See section 10.3.5.4, “Scheduler Attribute”, on page 783.

10.3.7 Recommended Manual Configuration

10.3.7.1 Set Scheduling Parameters

• Preempting jobs via suspension on the Cray causes problems. Do not attempt to use preemption via suspension. Make sure that the preempt_order scheduler parameter in PBS_HOME/sched_priv/sched_config does not include “S”. See section 4.8.3.8, “Preemption Methods”, on page 235.
• You must add the vntype resource to the “resources:” line in PBS_HOME/sched_priv/sched_config. If you want the scheduler to honor the following
Chapter 10  Integrations

resources, add them as well:
PBScrayhost
PBScraynid
PBScrayseg
accelerator_memory
accelerator_model

- Do not add the following resources to the “resources:” line:
PBScrayorder
nchunk

10.3.7.2 Setting Server and Queue Defaults

PBS does not support server or queue defaults for mpp* resources. You must replace any resources_default.mpp* settings as follows:

Table 10-4: How to Replace mpp* Server/Queue Defaults

<table>
<thead>
<tr>
<th>Old Resource Default</th>
<th>New Resource Default</th>
<th>Set New Value to Old Value of This:</th>
</tr>
</thead>
<tbody>
<tr>
<td>resources_default.mpp-width</td>
<td>default_chunk.nchunk</td>
<td>resources_default.mppwidth</td>
</tr>
<tr>
<td>resources_default.mpp-depth</td>
<td>default_chunk.ncpus</td>
<td>resources_default.mppdepth</td>
</tr>
<tr>
<td>resources_default.mppnppn (must replace with two settings)</td>
<td>default_chunk.mpiprocs</td>
<td>resources_default.mppnppn</td>
</tr>
<tr>
<td></td>
<td>default_chunk.ncpus</td>
<td>resources_default.mppdepth * mppnppn</td>
</tr>
<tr>
<td>resources_default.mpparch</td>
<td>default_chunk.arch</td>
<td>resources_default.mpparch</td>
</tr>
<tr>
<td>resources_default.mpphost</td>
<td>default_chunk.PBScrayhost</td>
<td>CLE 2.2: default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLE 3.0 and higher: resources_default.mpphost</td>
</tr>
</tbody>
</table>
### 10.3.7.3 Setting Server and Queue Minima and Maxima

PBS does not support server or queue minimum or maximum settings for `mpp*` resources. Instead of gating at the queue or server, you can do the following:

- Create a job-wide custom resource that can be requested by users or set by a hook. For example, to replace `mpparch`, create a string resource called `crayarch`, so users can request `crayarch` for their jobs.
- Write a `queuejob` hook that checks against minimum and maximum values for desired resources. See Chapter 6, "Hooks", on page 413.

#### 10.3.7.3.i Caveats for Replacing Resources Used for Gating

- There is no support for `resources_min.nchunk` and `resources_max.nchunk`. If set, their behavior is undefined.
- If you wish to set `resources_min.mpiprocs` or `resources_max.mpiprocs`, you must make sure that `mpiprocs` can be counted for each job chunk. If the job did not request `mpiprocs` with each chunk, the job must inherit `mpiprocs = 1` for each chunk.

```
Qmgr: s s default_chunk.mpiprocs = 1
```

#### 10.3.7.4 Using the nchunk Resource to Replace mppwidth

You can use the `nchunk` resource to replace the functionality of `mppwidth`. The `mppwidth` resource is used to indicate the number of PEs required by a job. Since a chunk is typically 1 PE, the `nchunk` resource can be used to indicate the number of PEs.
Chapter 10 Integrations

10.3.7.5 Setting Chunk Default For Vnode Type

Any job or reservation submitted with mpp* syntax is translated into select and place statements. Because the use of mpp* implies a request to use compute nodes, users should specify a vnode type of cray_compute. However, they may forget to do so. You can help out the users who are not used to specifying a vnode type to indicate that they are requesting compute nodes by setting default_chunk.vntype to cray_compute.

10.3.7.6 Allowing Scheduling on Cray Resources

If you want to be able to use the Cray resources such as accelerator_memory, accelerator_model, PBScrayhost, PBScrayseg, PBScraynid, vntype, etc., you must add them to the resources: line in PBS_HOME/sched_priv/config. Do not add PBScrayorder to the resources: line.

10.3.7.7 Using qstat Wrappers and Scripts

You must check any qstat wrappers or scripts. Either replace any mpp* resources, or modify the wrapper or script to use values in the job’s Submit_arguments attribute. This attribute contains the original job submission line.

10.3.7.8 Keeping Jobs Within One Host

To prevent jobs from being scheduled across multiple Cray hosts, you must limit jobs to a single value for PBScrayhost. Do the following:

- Set node_group_enable to True:
  Qmgr: s s node_group_enable=True
- Set node_group_key to PBScrayhost:
  Qmgr: s s node_group_key=PBScrayhost

If a job requests more resources than can be supplied from a single host, and the job does not specify a value for PBScrayhost, the job is scheduled across multiple Cray systems. To prevent this from happening, you can set the do_not_span_psets scheduler attribute to True. See section 10.3.5.4, “Scheduler Attribute”, on page 783.

10.3.7.8.i Caveat

You cannot use this technique on CLE 2.2, because PBScrayhost is set to “default” for all Cray hosts.
10.3.7.9 Allowing Scheduling on Nearby Vnodes

To help the scheduler place each job requiring more than one vnode on vnodes that are close to each other, make the scheduler sort the vnodes based on their values for PBScrayorder. The vnodes will be listed in the order that their nodes are listed in the Cray inventory. To do this, specify the following in PBS_HOME/sched_priv/config:

```
node_sort_key: "PBScrayorder LOW"
```

10.3.7.10 Allowing Users to Request Useful Groups of Nodes

Job submitters can use select and place to request the groups of vnodes they want. This is the same behavior that they were accustomed to getting with mppnodes. However, you must provide the tools. Users may need to group their nodes by the certain criteria, for example:

- Certain nodes are fast nodes
- Certain nodes share a required or useful characteristic
- Some combination of nodes gives the best performance for an application

For these cases, you can do either of the following:

- Create custom resources, and set them on each vnode so that the important characteristics of the vnode can be requested. For example, if a vnode is fast, create a custom string resource called “speed” and set it to fast on that vnode.
- Label each node with its important characteristics. For example, if a node is both fast and best for App1, give it two labels, fast, and BestForApp1. PBS creates custom Boolean resources called PBScraylabel_<label name> and sets them to True on the appropriate vnodes.

10.3.7.11 Allowing Users To Reserve $N$ NUMA Nodes Per Compute Node

You can allow a user to submit a Cray job with a PBS resource request that matches the Cray aprun -sn feature, which reserves some number of NUMA nodes per host. PBS can request the correct resources, but the user must translate the aprun syntax into PBS select and place syntax, and you must create the required custom resource for vnode selection, and set its value to create groupings of vnodes.

For example, a user wants 24 PEs in total, but only wants to use three NUMA nodes per host. The Cray syntax is aprun -sn 3 -n 24. If you have two compute nodes nid2 and nid5, each with 4 NUMA nodes, and each NUMA node has 4 PEs, this request can be satisfied by
Chapter 10 \hspace{1cm} Integrations

Taking the first 3 NUMA nodes on each host. The list of NUMA nodes is nid2_0, nid2_1, nid2_2, nid5_0, nid5_1, and nid5_2. In this example, the first 3 NUMA nodes on each host form a vnode grouping.

10.3.7.11.i Creating Custom Resource for Vnode Selection

You must create vnode groupings for the vnodes representing each compute node. Create groupings for each possible number of NUMA nodes. For example, for a compute node with 2 NUMA nodes, and therefore 2 vnodes, you create a grouping made up of 1 vnode, and a grouping of two vnodes. For a compute node with 4 NUMA nodes, groupings can be as large as 4 vnodes.

You create groupings by creating a custom string array resource, for example named \texttt{craysn}, and setting it to the list of grouping sizes that can contain that vnode. So, for example, the grouping that contains 2 vnodes is made up of NUMA nodes 0 and 1, and the \texttt{craysn} resource contains the value 2 on those vnodes.

One vnode always participates in all groupings on its compute node. One vnode only participates in the grouping that contains all the vnodes on the compute node.

The following table shows the values for \texttt{craysn} for a 4-NUMA-node compute node:

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
NUMA node & Value of \texttt{craysn} on This NUMA node’s Vnode \\
\hline
NUMA node 0 & 1, 2, 3, 4 \\
NUMA node 1 & 2, 3, 4 \\
NUMA node 2 & 3, 4 \\
NUMA node 3 & 4 \\
\hline
\end{tabular}
\end{table}

When a user requests a value of 3 for the \texttt{craysn} resource, the vnodes representing NUMA nodes 0, 1, and 2 are assigned to the job.

10.3.7.11.ii PBS Resource Request Reserving \textit{N} NUMA Nodes Per Host

To make a request equivalent to \texttt{aprun -sn 3 -n 24}, and match the compute node exclusive behavior of the Cray, the user can specify the following:

\texttt{qsub -lselect=24:ncpus=1:craysn=3 -lplace=exclhost}
Integrations

Chapter 10

10.3.7.12  Allowing Users To Reserve Specific NUMA Nodes

You can allow a user to submit a Cray job with a PBS resource request that matches the Cray `aprun -s1` feature, which reserves specific NUMA nodes on each compute node on which the job runs. In this case, the user does not request specific compute nodes.

PBS can request the correct resources, but the user must translate the `aprun` syntax into PBS select and place syntax. Information on translating the `aprun` syntax into select and place is in "Automatic Translation of mpp* Resource Requests", on page 265 of the PBS Professional User’s Guide.

For example, a user wants a total of 8 PEs, while using only NUMA node 1 on each compute node. The `aprun` syntax is `aprun -s1 1 -n 8`.

There are multiple options for how users place their jobs when specifying NUMA nodes.

10.3.7.12.i  Specifying Individual Vnodes

PBS automatically creates a custom string resource called `PBScrayseg`, and sets the value for each vnode to be the segment ordinal for the associated NUMA node. See "PBScrayseg" on page 783.

If the `aprun` syntax is the following:

```
aprun -s1 1 -n 8
```

An equivalent resource request for a PBS job is the following:

```
qsub -lselect=8:ncpus=1:PBScrayseg=1
```

10.3.7.12.ii  Specifying Multiple Vnodes

If `-s1` specifies a list of NUMA nodes, as in the following:

```
aprun -s1 0,1 -n 8
```

there are several ways to allow users to make the equivalent resource request.

- The PBS user can request separate chunks for each NUMA node:

```
qsub -lselect=4:ncpus=1:PBScrayseg=0 +4:ncpus=1:PBScrayseg=1
```

- If the user knows about the underlying hardware, the PBS resource request can take advantage of that. On a homogenous system with 2 NUMA nodes per compute node and 4 PEs per NUMA node, the PBS resource request can be the following:

```
qsub -lselect=8:ncpus=1 -lplace=pack
```

- You can create custom resources that group the underlying NUMA nodes together. Do
the following:

a. Create a string array resource named, for example, NUMAnodecombo

b. On every vnode, set NUMAnodecombo equal to all possible \texttt{s1} combinations that might be requested that include that vnode (NUMA node).

The following table shows the values for the \texttt{s1} combinations for a 4-NUMA-node compute node:

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{NUMA node} & \textbf{\texttt{s1} Combinations for NUMA node} \\
\hline
0 & 0,01,02,03,012,013,023,0123 \\
1 & 1,01,12,13,012,123,0123 \\
2 & 2,02,12,23,012,023,123,0123 \\
3 & 3,03,13,23,013,023,123,0123 \\
\hline
\end{tabular}
\end{center}

An equivalent select statement which uses this resource is the following:
\begin{verbatim}
qsub -lselect=8:ncpus=1:NUMAnodecombo=01 jobscript
\end{verbatim}

\begin{itemize}
\item You can use the vnsetofx resource:
\end{itemize}

You create placement sets by creating a custom string array resource, for example named \texttt{vnsetof\textless number of vnodes\textgreater}, and setting it to the list of placement sets containing \textless number of vnodes\textgreater that can contain that vnode.

On a 4 NUMA-node compute node represented by 4 vnodes, the complete list of possible placement sets is: \{0,1,2,3,01,02,03,12,13,23,012,013,023,123,0123,0123\}.

The following table shows the values for \texttt{vnsetof\textless number of vnodes\textgreater} for a 4-NUMA-node compute node:

\begin{center}
\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{NUMA Node} & \textbf{vnsetof1} & \textbf{vnsetof2} & \textbf{vnsetof3} & \textbf{vnsetof4} \\
\hline
NUMA node 0 & 0 & 01,02,03 & 012,013,023 & 0123 \\
NUMA node 1 & 1 & 01,12,13 & 012,013,123 & 0123 \\
NUMA node 2 & 2 & 02,12,23 & 012,023,123 & 0123 \\
\hline
\end{tabular}
\end{center}
10.3.7.12.iii Caution When Creating Resource

You must use the same resource string values on each vnode. “012” is not the same as “102” or “201”.

10.3.7.13 Allowing Users to Request Login Node Groups

If users need to request groups of both esLogin nodes and internal login nodes, do the following:

1. Create a new string value for the vntype resource, for example cray_compile.

2. Use qmgr to set the value for vntype on the vnodes representing esLogin nodes:

   qmgr -c “set node es_Login resources_available.vntype+="cray_compile""

3. Use qmgr to add cray_compile to the vnodes representing internal login nodes.
   (resources_available.vntype is automatically set to cray_login.)

   qmgr -c “set node internal_Login resources_available.vntype +="cray_compile""

If you use pbsnodes -av to check resources_available.vntype for internal_Login, it now looks like this:

   resources_available.vntype=cray_login,cray_compile

10.3.7.14 Allowing Users to Request Interlagos Hardware

You can provide users with a way to request or avoid Interlagos hardware. We recommend that you use xtprocadmin to set a label called “interlagos” on each compute node that uses Interlagos hardware. This way, PBS automatically creates the Boolean PBScraylabel_interlagos and sets it to True on each corresponding vnode.
Chapter 10  Integrations

If you cannot set the label, you can do the following:

1. On each vnode that belongs to the same host, edit the `server_priv/resourcedef` file and add a Boolean to represent whether or not a vnode uses Interlagos hardware:
   ```bash
   PBScraylabel_interlagos type=boolean flag=h
   ```


3. Use `qmgr` to set the resource on the corresponding vnodes:
   ```bash
   qmgr: set node <vnode name>
   resources_available.PBScraylabel_interlagos = True
   ```

Users can then request or avoid this resource using `PBScraylabel_interlagos=True` or `PBScraylabel_interlagos=False`. For example:

```bash
qsub -lselect=3:ncpus=2:PBScraylabel_interlagos=true myjob
```

10.3.7.15  Set ALPS Reservation Release Timeout

The `$alps_release_timeout` parameter specifies the amount of time that PBS tries to release an ALPS reservation before giving up. After this amount of time has passed, PBS stops trying to release the ALPS reservation, the job exits, and the job’s resources are released. PBS sends a HUP to the MoM so that she rereads the ALPS inventory to get the current available ALPS resources.

We recommend that you set the value for the `$alps_release_timeout` MoM parameter to a value greater than the value of the `suspectbegin` Cray node health variable. You want to allow the Cray node health check to reach its timeout before PBS gives up on trying to release the reservation.

The default for `$alps_release_timeout` is 600 seconds.

10.3.7.16  Enable Local Copy

If your site has disabled the use of remote operation functions ("r" commands) and output cannot be returned for jobs running on compute nodes, enable the use of the `cp` command by adding `$usecp` to the `$PBS_HOME/mom_priv/config` file on each login node.

10.3.7.17  Do Not Set `kill_delay` to Less than Ten Seconds

The default for `kill_delay` is 10 seconds. It is not recommended that you set it to less than 10 seconds.

```bash
<server host>: # qmgr -c "set queue workq kill_delay=<delay>"
```
10.3.7.18 Enable mpp Resource Translation Hook

You need to perform this step only if the PBS server is on a non-Cray host. Enable the mpp resource translation hook by editing the server_priv/hooks/PBS_translate_mpp.HK file, and setting "enabled" to True.

10.3.8 Viewing Cray Information

10.3.8.1 Listing Vnodes

Each vnode appears only once in the output of any command that lists all vnodes, such as pbs_statnode(), pbsnodes -av, or qmgr -c "list nodes @default".

10.3.8.2 Contents of Vnode Mom Attribute

When multiple login nodes are defined, each vnode representing a compute node lists every reporting login node in its Mom attribute. The Mom attribute lists multiple fully-qualified host names in a comma-separated list format.

For example, MoM1 reports compute nodes with node_id 1 through 4, and MoM2 reports compute nodes with node_id 3 through 6. In this case, the vnodes representing compute nodes with node_id 1 and 2 list MoM1 in the Mom attribute, vnodes representing compute nodes with node_id 3 and 4 list MoM1 and MoM2 in the Mom attribute, and vnodes representing compute nodes with node_id 5 and 6 list MoM2 in the Mom attribute.

10.3.8.3 Viewing Job and Reservation Information

PBS translates all job and reservation mpp* syntax into select and place statements. See "Automatic Translation of mpp* Resource Requests", on page 265 of the PBS Professional User's Guide. The job's Submit_arguments attribute contains the original submission arguments. When you use the qstat -f, -x, or -H command to view job attributes, you see the following:

- The Resource_List attribute contains the translated select and place syntax, instead of the mpp* syntax
- The Submit_arguments attribute contains the original resource request

When you use the pbs_rstat -F command to view reservation attributes, you see the following:

- The Resource_List attribute contains the translated select and place syntax, instead of the mpp* syntax
10.3.8.4 Viewing Vnode Information

Each vnode’s jobs attribute lists the jobs that have processes executing on that vnode. Jobs launched from an internal login node, requesting a vntype of cray_compute only, are not listed in the internal login node’s vnode’s jobs attribute. Jobs that are actually running on a login node, which requested a vntype of cray_login, do appear in the login node’s vnode’s jobs attribute.

You can view vnode attributes using the pbsnodes -av command.

If esLogin and internal login nodes are grouped by adding a string such as cray_compile to the vntype resource, the pbs_rstat -F command shows the following:

```
resources_available.vntype=cray_login,cray_compile
```

10.3.9 Jobs on the Cray

Cray users can submit jobs using the mpp* syntax; this is translated into select and place statements. See "Automatic Translation of mpp* Resource Requests", on page 265 of the PBS Professional User’s Guide for a translation description.

10.3.9.1 Effect on Jobs of Stopping and Starting Vnodes

If a job is launched from a login node, and the MoM goes down, the impact on the job depends on the command-line options specified when the MoM is restarted. See “pbs_mom” on page 58 of the PBS Professional Reference Guide.

If a job is launched from a login node, and the login node goes down, the job does not continue to run.

If there are multiple login nodes, and one login node or its MoM goes down, jobs that were launched from other login nodes are not affected.

10.3.9.2 Resource Request Translation

When a PBS job or reservation is submitted using the mpp* syntax, PBS translates the mpp* resource request into PBS resources. The translation uses the rules described in "Automatic Translation of mpp* Resource Requests", on page 265 of the PBS Professional User’s Guide.

10.3.9.3 Resource Accounting

Comprehensive System Accounting (CSA) runs on the compute nodes, under the control of the Cray system. PBS performs resource accounting on the login nodes, under the control of their MoMs.
10.3.9.3.i Using Comprehensive System Accounting

If CSA is enabled, PBS can request the kernel to write user job accounting data to accounting records. These records can then be used to produce reports for the user.

If PBS finds the CSA shared object libraries, and CSA is enabled, PBS can cause a workload management record to be written for each job. If MoM is configured for CSA support, MoM can issue CSA workload management record requests to the kernel. The kernel writes workload management accounting records associated with the PBS job to the system-wide process accounting file. The default for this file is /var/csa/day/pacct.

10.3.9.3.ii CSA Configuration Parameter

```
pbs_accounting_workload_mgmt <value>
```

MoM configuration parameter. Controls whether CSA accounting is enabled. The name does not start with a dollar sign. If set to "1", "on", or "true", CSA accounting is enabled. If set to "0", "off", or "false", CSA accounting is disabled. Values are case-insensitive. Default: "true"; enabled.

10.3.9.3.iii Requirements for CSA

PBS supports CSA on Cray machines. CSA requires CSA support Linux kernel modules. On the supported platforms, the PBS MoM is CSA-enabled. If CSA workload management and user job accounting are available, PBS can use them.

10.3.9.3.iv Configuring MoM for CSA

CSA support is specified in the `pbs_accounting_workload_mgmt` line in MoM’s Version 1 configuration file. CSA support is enabled by default; you must explicitly disable it if you want it disabled. If the `pbs_accounting_workload_mgmt` line is absent, CSA is still enabled.

To disable CSA support, modify `$PBS_HOME/mom_priv/config`, by setting `pbs_accounting_workload_mgmt` to `false`, `off`, or `0`.

To enable CSA support, either remove the `pbs_accounting_workload_mgmt` line, or set it to `true`, `on`, or `1`.

After modifying the MoM config file, either restart `pbs_mom` or send it SIGHUP.

10.3.9.3.v Enabling Kernel CSA Support

In order for CSA user job accounting and workload management accounting requests to be acted on by the kernel, you need to make sure that the parameters `CSA_START` and `WKMG_START` in the `/etc/csa.conf` configuration file are set to "on" and that the system reflects this. You can check this by running the command:

```
csaswitch -c status
```

PBS Professional 12.2 Administrator’s Guide 799
Chapter 10 Integrations

To set CSA_START to on, use the command:

```bash
     csaswitch -c on -n csa
```

To set WKMG_START to on, use:

```bash
     csaswitch -c on -n wkmg
```

Alternatively, you can use the CSA startup script /etc/init.d/csa with the desired argument (on/off); see the system's man page for csaswitch and how it is used in the /etc/init.d/csa startup script.

10.3.10 Resource Restrictions and Deprecations

PBS translates only the following mpp* resources into select and place syntax:

- mppwidth
- mppdepth
- mppnppn
- mppmem
- mpparch
- mpphost
- mpplabels
- mppnodes

The mpp* syntax is deprecated. See section 1.3, “Deprecations and Removals”, on page 9.

The following are not supported, and if set, behavior is undefined.

- resources_min.nchunk
- resources_max.nchunk.

10.3.11 Caveats and Advice

10.3.11.1 Configure Cray MoMs According to Rules

- When configuring a Cray MoM, follow the rules in section 3.5.2.2, “Configuring Multi-vnode Machines without cpusets”, on page 52. Note that since the vnode sharing attribute must be set using the pbs_mom -s insert command, it is not recommended to set the sharing attribute on a Cray vnode.
10.3.11.2  Do Not Preempt Jobs via Suspension

- Preempting jobs via suspension on the Cray causes problems. Do not attempt to use preemption via suspension. Make sure that the preempt_order scheduler parameter in PBS_HOME/sched_priv/sched_config does not include “S”. See section 4.8.33.8, “Preemption Methods”, on page 235.

10.3.11.3  Do Not Suspend Jobs on Cray

- Do not attempt to use qsig -s suspend on the Cray. Attempting to suspend a job on the Cray will cause errors.

10.3.11.4  Vnode Definition Files Not Recommended

Using vnode definition files on a Cray is not recommended. Use qmgr instead.

Any attribute and resource settings for a specific vnode in a vnode definition file cause PBS to believe that that vnode is still usable. If PBS reads the Cray inventory, and a vnode is not listed in the inventory, but it is listed in a vnode definition file, the vnode is not marked as stale. This will cause a problem when the scheduler tries to schedule jobs onto this vnode.

If you create a vnode definition file for a vnode that has more than one MoM, you must make sure that the files are consistent on all of the MoMs that manage the vnode.

Settings in a vnode definition file override those from inventory. Settings in qmgr override both vnode definition files and inventory. Be careful not to overwrite information from the inventory when creating a vnode definition file. For example, if the vntype resource for a vnode is set to cray_login when PBS reads the inventory, and it is set to cray_compile in a vnode definition file, the value for vntype becomes cray_compile only.

10.3.11.5  Use Correct Name When Creating Vnode

When creating a vnode to represent a login node, use the short name returned by the gethostname command on the login node. For example, if gethostname returns HostA, do the following:

Qmgr: create node HostA

If you create a vnode with a different name from the short name returned by gethostname, the following happens:

- MoM creates a vnode whose name is the short name returned by gethostname
- The vnode you created is not recognized by MoM, and is marked stale
10.3.11.6 Deleting Vnodes on Cray

You can delete a natural vnode only if no other vnodes list this vnode in their Mom attributes. In order to delete a natural vnode which is listed in another vnode’s Mom attribute, you must first delete the vnode with this vnode in its Mom attribute.

After removing a vnode that is managed by more than one MoM, you must HUP all of the managing MoMs, otherwise the vnode is not marked stale by the server.

10.3.11.7 Do Not Make Vnode Definitions Additive

On a Cray MoM, the $vnode_def_additive parameter in PBS_HOME/mom_priv/config is set to False or 0 by default. Do not unset or change the setting of the $vnode_def_additive parameter.

10.3.11.8 For CLE 2.2, No Value for mpp_host

On CLE 2.2, with BASIL 1.1, there is no value for mpp_host. Therefore, you cannot use this to distinguish between Cray hosts.

10.3.11.9 Do Not Use configrm

It is not recommended to use the configrm pbs_tclsh call.

10.3.11.10 Using Gating Values As Defaults

For most resources, if the job does not request the resource, and no server or queue defaults are set, the job inherits the maximum gating value for the resource. If this is set at the queue, the queue value of resources_max.<resource> is used. If this is set only at the server, the job inherits the value set at the server. However, for mpp* resources, the job does not inherit the gating value. For example, if the job does not request mppnppn, and no defaults are set at the server and queue, but resources_max.mppnppn is set at the queue, the job does not inherit the queue’s value.

10.3.11.11 Marking Cray Vnodes Offline

You can use the qmgr command to individually mark each vnode representing a compute node offline. This is independent of the vnodes representing login nodes.
10.3.11.12 Do Not Use PBS-reserved Resource Names

Do not create resources with names that could be used by PBS to create a Cray resource equivalent. For example, do not create a resource with the name PBS\textit{craylabel\_small}.

10.3.11.13 Regularly Re-reading Inventory

If you want PBS to re-read the Cray inventory on a regular schedule, you can set up a \texttt{cron} job to HUP the Cray MoM(s).

10.3.11.14 Do Not Schedule on PBS\textit{crayorder}

Do not add PBS\textit{crayorder} to the \texttt{resources:} line.
10.3.11.15 Fewer Chunks for Shorter Scheduling Cycle

The more chunks in each translated job request, the longer the scheduling cycle takes. Jobs that request a value for mppnppn or ncpus effectively direct PBS to use the size of mppnppn or ncpus as the value for ncpus for each chunk, thus dividing the number of chunks by mpp-nppn or ncpus.

Example 10-1: Comparison of larger vs. smaller chunk size and the effect on scheduling time:

Submit job with chunk size 1 and 8544 chunks:

```
qsub -lmppwidth=8544 job
```

Job's Resource_List:

- Resource_List.mppwidth = 8544
- Resource_List.ncpus = 8544
- Resource_List.place = free
- Resource_List.select = 8544:vpctype=cray_compute
- Submit arguments = -lmppwidth=8544 job

Scheduling took 6 seconds:

```
12/05/2011 16:46:10;0080;pbs_sched;Job;23.example;Considering job to run
12/05/2011 16:46:16;0040;pbs_sched;Job;23.example;Job run
```

Submit job with chunk size 8 and 1068 chunks:

```
qsub -lmppwidth=8544,mppnppn=8 job
```

Job's Resource_List:

- Resource_List.mpiprocs = 8544
- Resource_List.mppnppn = 8
- Resource_List.mppwidth = 8544
- Resource_List.ncpus = 8544
- Resource_List.place = scatter
- Resource_List.select = 1068:ncpus=8:mpiprocs=8:vpctype=cray_compute

Scheduling took 1 second:

```
12/05/2011 16:54:38;0080;pbs_sched;Job;24.example;Considering job to run
12/05/2011 16:54:39;0040;pbs_sched;Job;24.example;Job run
```

804 PBS Professional 12.2 Administrator's Guide
To speed up scheduling, you may want to write a submission hook that assigns each job a value for mppnppn or ncpus. We recommend that this value be the value for ncpus for a vnode or for a compute node.

Example 10-2: Using submission hook, assign value for mppnppn:

Submit job with chunk size 1 and 8544 chunks:

```
qsub -lmppwidth=8544 job
```

Job’s Resource_List:

- Resource_List.mpi_procs = 8544
- Resource_List.mppnpn = 8
- Resource_List.mppwidth = 8544
- Resource_List.ncpus = 8544
- Resource_List.place = scatter
- Resource_List.select = 1068:ncpus=8:mpiprocs=8:vntype=cray_compute

Submit_arguments = -lmppwidth=8544 job

Scheduling took 1 second:

12/05/2011 16:35:04;0080;pbs_sched;Job;20.example;Considering job to run
12/05/2011 16:35:05;0040;pbs_sched;Job;20.example;Job run

Submit job with chunk size 8 and 1068 chunks:

```
qsub -lmppwidth=8544,mppnpn=8 job
```

Job’s Resource_List:

- Resource_List.mpi_procs = 8544
- Resource_List.mppnpn = 8
- Resource_List.mppwidth = 8544
- Resource_List.ncpus = 8544
- Resource_List.place = scatter
- Resource_List.select = 1068:ncpus=8:mpiprocs=8:vntype=cray_compute

Submit_arguments = -lmppwidth=8544,mppnpn=8 job

Scheduling took 1 second:

12/05/2011 16:35:48;0080;pbs_sched;Job;21.example;Considering job to run
12/05/2011 16:35:49;0040;pbs_sched;Job;21.example;Job run
10.3.11.15 Caveats:

If you are on a heterogeneous system, forcing mppnppn to be set for all requests can cause jobs to wait for particular vnodes and their associated resources, where these jobs would have been able to run across many different-sized vnodes.

Instead, you can use a job submission hook that rejects jobs that don’t request values for either mppnppn or ncpus.

10.3.11.16 No cput or mem for Compute Node Jobs

PBS does not report cput or mem for jobs running on a Cray compute node. Note that this prevents cput from being used for checkpointing on a Cray.

10.3.11.17 Set PATH Correctly

PATH must be included in the pbs_environment file. The PATH value is passed on to batch jobs. To maintain security, it is important that PATH be restricted to known, safe directories. Do not include "." in PATH.

10.3.12 Errors and Logging

10.3.12.1 Creating Custom Resources

When a custom resource is created for a Cray vnode, the server logs a message containing the resource name and type, and the vnode name. This is logged at event class 0x080.

If a custom resource can’t be created, the following error message is printed in the server log:

```
error: resource <name> for vnode <name> cannot be defined
```

10.3.12.2 Job Requests More Than Available

If do_not_span_psets is set to True, and a job requests more resources than are available in one placement set, the following happens:

- The job's comment is set to the following:
  
  “Not Running: can't fit in the largest placement set, and can't span placement sets”

- The following message is printed to the scheduler’s log:
  
  “Can't fit in the largest placement set, and can't span placement sets”
10.3.12.3 Invalid Cray Requests

It is possible to create a select and place statement that meets the requirements of PBS but not of the Cray. The Cray width and depth values cannot be calculated from ncpus and mpiprocs values. For example, if ncpus is 2 and mpiprocs is 4, the depth value is calculated by dividing ncpus by mpiprocs, and is one-half. This is not a valid depth value for Cray. When a select statement does not meet Cray requirements, and the Cray reservation fails, the following error message is printed in MoM’s log, at log event class 0x080:

Fatal MPP reservation error preparing request

10.3.12.4 If ompthreads and ncpus Not Equal

If the value of ompthreads does not match the value of ncpus when PBS is constructing exec_vnode for a job, the following is printed in the MoM log, at event class 0x080:

“ompthreads <value> does not match ncpus <value>”

10.3.12.5 All Requested mppnodes Not Found

If mppnodes are requested, but there are no vnodes that match the requested mppnodes (i.e. 0% of the mppnodes list is found), the job or reservation is rejected with the following message:

“The following error was encountered: No matching vnodes for the given mppnodes <mppnodes>”

A log message is printed to the server log at event class 0x0004:

“translate mpp: ERROR: could not find matching vnodes for the given mppnodes <mppnodes (as input)>”

10.3.12.6 Some Requested mppnodes Not Found

If mppnodes are requested, and only some of the mppnodes are found to match the vnodes, then the job/reservation is accepted, but the following is printed in the server log at event class 0x0004:

“translate mpp: could not find matching vnodes for these given mppnodes [<comma separated list of mppnodes>]”

The job may or may not run depending on whether the vnodes that were matched up to the requested mppnodes have enough resources for the job.
Chapter 10 Integrations

10.3.12.7 Bad mppnodes Range

If the resource request specifies an mppnodes range with the value on the right hand side of the range less than or equal to the value on the left hand side of the range, the job or reservation is rejected with the following message:

The following error was encountered:

Bad range '<range>', the first number (<left_side>) must be less than the second number (<right_side>)

A log message is printed to the server log at event class 0x0004:

"translate mpp: ERROR: bad range '<range>', the first number (<left_side>) must be less than the second number (<right_side>)"

10.3.12.8 Resource Request Containing Both mpp* and select/place

If a resource request contains both mpp* and select/place, the job or reservation is rejected, and the following error is printed:

"The following error was encountered:

mpp resources cannot be used with "select" or "place"
"

10.4 Support for SGI

10.4.1 Briefly, How PBS Manages Cpusets

PBS provides an enhanced PBS MoM called pbs_mom.cpuset, which is designed to manage the cpusets on a cpusetted machine. The standard PBS MoM, called pbs_mom.standard, can manage a machine with cpusets, but in this case PBS and the jobs it manages will not create or otherwise make use of cpusets.

On the cpusetted machines managed by pbs_mom.cpuset listed in section 10.4.8.1, "Generation of Placement Set Information", on page 812, PBS automatically examines the topology of the machine, and creates vnodes to represent subsets of the machine. PBS also organizes the machine’s vnodes into placement sets. When PBS runs a job on an execution host managed by pbs_mom.cpuset, the PBS MoM creates the cpuset in which the job runs, and destroys the cpuset after the job is finished. How nice is that?
10.4.2 Cpusets and Vnodes

The cpuset PBS MoM represents a machine as a set of vnodes. Each vnode is visible via commands such as `pbsnodes`. Each vnode must have its own logical memory pool, so you get one vnode per logical memory pool. All of the vnodes on one Altix are managed by one instance of `pbs_mom.cpuset`.

A cpuset is a group of CPUs and memory nodes around which an inescapable wall has been placed. The OS manages a cpuset so that processes executing within the cpuset are typically confined to use only the resources defined by the cpuset.

10.4.3 Requirements for Managing Cpusets

If you want PBS to manage the cpusets on a machine, all of the following must be true:

- You run `pbs_mom.cpuset`, instead of the standard `pbs_mom`.
- The machine runs a supported version of the SGI ProPack library or SGI Performance Suite.
- You use the PBS start/stop script to start the machine. In order to use `pbs_mom.cpuset` on an Altix, you need a vnode definitions file, which contains all the information about the machine’s vnodes and their resources. This is used by PBS for scheduling jobs; the scheduler uses the information to create placement sets. Each machine may have a different topology, depending on how it is wired. PBS creates this file automatically when the PBS start/stop script runs.
- If the machine is an ICE, the file named `/etc/sgi-compute-node-release` must be present.

10.4.4 Where to Use Cpusets

Use PBS to manage your cpusets wherever you want jobs to be fenced into their own CPUs and memory. This technique is common on Altix and UV 1000 machines. This can also be useful on other machines, such as ICE, depending on the individual machine.

10.4.5 Creation of Vnode Definitions File

PBS creates the vnode definitions file if it detects that `pbs_mom.cpuset` has been copied to `pbs_mom`. The vnode definitions file is generated automatically by the PBS start/stop script when it runs a vnode generation script. The start/stop script affects only the host on which it is run.
On Altix and UV, the script that generates the vnode definitions is `$PBS_EXEC/lib/init.d/sgigenvnodelist.awk`.

On ICE, there are two scripts, and each produces a file that is used for vnode definitions. These scripts are named `$PBS_EXEC/lib/init.d/sgiICEvnode.sh` and `$PBS_EXEC/lib/init.d/sgiICEplacement.sh`.

The vnode generation script produces vnode definitions only for the host on which it is run. MoM sets the sharing attribute for each vnode as follows:

- On Altix and UV, MoM sets the `sharing` attribute for the natural vnode to `ignore_excl`.
- On Altix and UV, MoM sets the `sharing` attribute for all other vnodes to `default_excl`.
- On ICE, MoM sets the `sharing` attribute for each vnode to `default_shared`.

### 10.4.5.1 Changing Vnode Definitions

You can change the vnode definitions in the following ways:

- You can use the `pbs_mom -s insert` command to add vnode definitions that override those in the PBS-created vnode definition file. See section 3.5.3, “Creating Version 2 MoM Configuration Files”, on page 53.
- The vnode generation script is designed so that you can use modified copies to generate desired vnode definitions. Do not modify the original script; make copies, modify them as you wish, and insert the resulting vnode definitions using the `pbs_mom -s` option.

### 10.4.5.2 Caveats for Creating Vnodes

Do not attempt to configure more than one vnode per logical memory pool. Your jobs will not run correctly.

### 10.4.6 Using Boot Cpusets

A boot cpuset contains one or more CPUs and memory boards and is used to restrict the default placement of system processes, including login. If defined, the boot cpuset contains CPU 0.

PBS does not run in the boot cpuset unless placed there by you.
10.4.7  Creation of Cpusets by MoM

On each cpuset machine, MoM creates a cpuset under the root cpuset called PBSPro, containing all of the CPUs and memory that MoM is allowed to use. Whenever MoM creates a cpuset for a job, that cpuset is created only from CPUs and memory in the PBSPro cpuset.

The MoM uses only those CPUs that are in sets belonging to PBS; she will not use any CPUs in sets that do not belong to PBS. Sets that do not belong to PBS include the boot cpuset, if it exists.

At startup, MoM checks for the following CPUs to exclude:

- Any CPU in an existing, non-root, non-PBS cpuset
- The cpuset_create_flags flag set to CPUSET_CPU_EXCLUSIVE in the mom_priv/config file, which specifies that CPU 0 will not be used by PBS

If she finds CPUs to exclude, MoM does the following:

- Excludes that CPU from the top set /dev/cpuset/PBSPro
- Creates the top set with mem_exclusive set to False

If she does not find CPUs to exclude, she creates the top set using all CPUs and with mem_exclusive set to True.

MoM considers CPUs for inclusion in PBSPro as they are grouped by vnode. So for example if one CPU in a vnode is in use at startup, she excludes from PBSPro all of the CPUs in that vnode.

When MoM creates job cpusets, she does not set the CPU or memory exclusive flags. PBS manages the exclusivity on these cpusets.

10.4.7.1  Including and Excluding CPU 0

By default, CPU 0 is excluded from the PBSPro cpuset. The CPUSET_CPU_EXCLUSIVE flag prevents CPU 0 from being used by the MoM in the creation of job cpusets. This flag is set by default.

In order to include CPU 0, change the MoM configuration file line to this:

```bash
cpuset_create_flags 0
```

10.4.7.2  Manual Creation of Cpusets Not Managed by PBS

You may wish to create cpusets that will not be managed by PBS. If you want to reserve some CPUs to be used by non-PBS processes, you can create a cpuset for them.
If you have not started PBS, create these cpusets before starting PBS. If you have started PBS, requeue any jobs, stop PBS, create your cpuset(s), then restart PBS.

For example, to create a boot cpuset, requeue any jobs, stop the PBS MoM, make the change to the boot cpuset, then restart the PBS MoM. The PBSPro cpuset is then created at startup and will not include CPUs and memory in the boot cpuset.

10.4.7.3 Caveats for Creating Cpusets

10.4.7.3.i Number of CPUs Must Be Accurate

When you configure vnodes on an Altix, you can tell PBS that there are up to the actual number of CPUs in each vnode, but no more. The Altix assigns real hardware when it creates a cpuset. It tries to create cpusets containing the number of CPUs that you specified to PBS. PBS will try to assign all the CPUs in a cpuset to a job requiring that number. So if you tell PBS that a cpuset contains more than the number of actual CPUs, then when the Altix tries to create a cpuset for that job, it will fail and the job won’t run.

For example, if a vnode has 2 physical CPUs, you can tell PBS that there are 0, 1, or 2 CPUs, but no more. If you tell PBS that the vnode has 4 CPUs, the Altix will not be able to create the cpuset since only 2 CPUs are available.

It is not recommended to change the number of CPUs reported by MoM.

10.4.8 Placement Sets on SGI Machines

PBS creates placement sets using resources such as CPUs or distance between vnodes which are read from a topology file. For information on placement sets, see section 4.8.32, “Placement Sets”, on page 211.

When the scheduler chooses a placement set for a job, PBS creates a cpuset for that job taken from the CPUs and memory from that placement set. Each cpuset may be made up of one or more vnodes, depending on the requirements of the job. The scheduler orders the vnodes in the placement set according to the specified vnode sorting method, and uses vnodes in that order until it has enough CPUs and memory for the job. These are the vnodes used in the job’s cpuset.

10.4.8.1 Generation of Placement Set Information

PBS automatically generates the placement information for vnodes on the following machines:

- Shared-memory Altix running supported versions of ProPack
- Shared-memory Altix running the SGI Performance Suite
On an SGI ICE, UV 100, and UV 1000, PBS automatically generates vnode definitions, but not placement information. On a UV, this would result in a very large number of placement sets. You can create placement sets for these machines by hand.

10.4.9 Configuring a Cpusetted Machine

The rules for configuring a cpusetted machine are different from the rules for a non-cpusetted machine. Use the rules in section 3.5.2.3, “Configuring Machines with Cpusets”, on page 52.

10.4.10 Special Treatment for SGI ICE

The SGI ICE is treated differently in the following fashion:

- PBS uses the file named /etc/sgi-compute-node-release to identify the machine as an ICE
- PBS uses a different script to generate the vnode definitions; this script is named PBS_EXEC/lib/init.d/sgiICEvnode.sh
- The sharing attribute for each vnode is set to default_shared

10.4.10.1 Requirements for SGI ICE

Make sure that the file /etc/sgi-compute-node-release is present.

10.4.10.2 Vnodes on the ICE

The PBS vnode definition script reads the ICE topology information, and configures one vnode per logical memory pool. For many ICE machines, this results in a single vnode per MoM.

Vnodes on an SGI ICE are not allocated exclusively. They are shared, like vnodes in a normal cluster.

10.4.11 Special Treatment for SGI UV

If you are using memory enforcement on an SGI UV, set the MEMACCTD_FALLBACK flag to 0. This flag is passed to memacctd. memacctd is in userspace, and calls the SGI numa.ko kernel module from there.

It is usually sourced from an /etc/sysconfig file, so to change it at PBS installation, edit the sysconfig file, and restart memacctd, using the following:

/etc/init.d/memacctd restart
We recommend that you `chkconfig` it on.

### 10.4.12 Viewing Cpuset Information

To verify which CPUs are included in a cpuset created by PBS, use:

```bash
  cpuset -d <set name> | egrep cpus
```

This will work both from within a job and outside it.

The `alt_id` job attribute has the form `cpuset=<name>`. `<name>` is the name of the cpuset, which is the directory name or the full path to the cpuset. The value in `$PBS_JOBID`, the job’s ID, is the directory name.

### 10.4.13 Suspended Jobs and Cpusets

On a suspend request, the cpuset MoM moves job processes to the global cpuset, then restores them upon restart.

### 10.4.14 Cpuset-specific Configuration Parameters

The following configuration parameters in the version 1 MoM configuration file are specific to cpusets:

- **cpuset_create_flags <flags>**
  - Allowed values: `CPUSET_CPU_EXCLUSIVE | 0`
  - Default: `CPUSET_CPU_EXCLUSIVE`

- **cpuset_destroy_delay <delay>**
  - MoM waits `delay` seconds before destroying a cpuset of a just-completed job.
  - This allows processes time to finish.
  - Default: `0`.
  - Format: Integer.
  - Example:
    ```bash
    cpuset_destroy_delay 10
    ```
10.4.15 Switching From Standard MoM to Cpuset MoM on Altix

Here are the steps:

1. Using qmgr, delete the vnode run by the MoM to be switched:
   ```bash
   Qmgr: delete node foo
   ```
2. Stop PBS:
   ```bash
   /etc/init.d/pbs stop
   ```
3. Change directory to PBS_EXEC
4. Copy cpuset MoM to MoM:
   ```bash
   cp pbs_mom.cpuset pbs_mom
   ```
5. Start PBS:
   ```bash
   /etc/init.d/pbs start
   ```
6. Using qmgr, create natural vnode:
   ```bash
   Qmgr: create node foo
   ```
7. Optional: create a modified vnode definitions file with any changes that you made previously via qmgr. Use the `pbs_mom -s insert` command to add it.

Make sure that `mem`, `vmem`, `ncpus` or `sharing` are unset on the new vnode.

For information on using qmgr, see “qmgr” on page 150 of the PBS Professional Reference Guide.
10.4.16 Switching From Cpuset MoM to Standard MoM on Altix

If you switch from the cpusetted MoM to the standard MoM on the Altix, you’ll need to remove any vnode definition files you added that contain information dependent on the automatically-generated ones:

1. List your own vnode definitions files, and remove each file you added:
   
   ```
   pbs_mom -s list | while read def
   do
     pbs_mom -s remove $def
   done
   ```

2. Add new configuration files with any information you need:
   
   ```
   pbs_mom -s insert <new script>
   ```

3. Stop the MoM:
   
   ```
   <path to start/stop script>/pbs stop
   ```

4. Remove the now stale vnodes:
   
   ```
   Qmgr: delete node <vnode name>
   ```

5. Start the MoM:
   
   ```
   <path to start/stop script>/pbs start
   ```

10.4.17 Troubleshooting Cpusets

The cpuset-enabled MoM may occasionally encounter errors during startup from which it cannot recover without help. If pbs_mom was started without the -p flag, you may see this:

"/PBSPro hierarchy cleanup failed in <dir> - restart pbs_mom with '-p'"

where `<dir>` is one of /PBSPro, /PBSPro/shared, or /PBSPro/suspended. If this occurs, try restarting pbs_mom with the -p flag. If this succeeds, no further action is necessary to fix this problem. However, it is possible that if pbs_mom is started with the -p flag, you may then see any of these messages:

"cpuset_query for / failed - manual intervention is needed"
"/PBSPro query failed - manual intervention is needed"
"/PBSPro cpuset_getmems failed - manual intervention is needed"
In this case, there is likely to be something wrong with the PBSPro cpuset hierarchy. First, use the\texttt{cpuset(1)} utility to test it:

\begin{verbatim}
  # cpuset -s /PBSPro -r | while read set
do
    cpuset -d $set > /dev/null
  done
\end{verbatim}

If\texttt{cpuset} detects no problems, no output is expected. If a problem is seen, expect output of the form

\begin{verbatim}
  cpuset </badset> query failed
  /badset: Unknown error
\end{verbatim}

In this case, try to remove the offending cpuset by hand, using the\texttt{cpuset(1)} utility:

\begin{verbatim}
  # cpuset -x badset
  cpuset <badset> removed.
\end{verbatim}

This may fail because the named cpuset contains other cpusets, because tasks are still running attached to the named set, or other unanticipated reasons. If the set has subsets:

\begin{verbatim}
  # cpuset -x nonempty
  cpuset <nonempty> remove failed
  /nonempty: Device or resource busy
\end{verbatim}

first remove any cpusets it contains:

\begin{verbatim}
  # cpuset -s nonempty -r
  /nonempty
  /nonempty/subset
  ...

  # cpuset -s nonempty -r | tac | while read set
do
    cpuset -x $set
  done
  ...
  cpuset </nonempty/subset> removed.
  cpuset </nonempty> removed.
\end{verbatim}

Note that output is previous output, reversed.
Chapter 10 Integrations

If the set has processes that are still attached,

```bash
# cpuset -x busy
cpuset <busy> remove failed
/busy: Device or resource busy
```

you can choose either to kill off the processes:

```bash
# kill `cpuset -p busy`
# cpuset -x busy

```

cpuset <busy> removed.

or wait for them to exit. In the latter case, be sure to restart pbs_mom using the -p flag to prevent it from terminating the running processes.

Finally, note that if removing a cpuset with `cpuset -x` should fail, you may also try to remove it with `rmdir(1)`, provided you take care to prepend the cpuset file system mount point first. For example,

```bash
# mount | egrep cpuset
cuset on /dev/cpuset type cpuset (rw)
# find /dev/cpuset/nonempty -type d -print | tac | while read set
do
   rmdir $set
done
```

10.4.18 Using Comprehensive System Accounting

PBS support for CSA on SGI systems is no longer available. The CSA functionality for SGI systems has been removed from PBS.

10.5 Support for Globus

Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus.

10.6 Support for Hyper-Threading

On Linux machines that have Hyper-Threading Technology, PBS can end up reporting and using the number of logical processors, instead of the number of physical CPUs, as the value for `resources_available.ncpus`.

818 PBS Professional 12.2 Administrator’s Guide
PBS does not control how CPUs are allocated to processes within a job. That is handled by the OS kernel.

**10.6.1 Linux Machines with HTT**

On Linux, PBS uses the number of CPUs shown in `/proc/cpuinfo`. If the CPUs are hyper-threaded and hyper-threading is enabled, the number of virtual and physical CPUs is different.

**10.6.2 Windows Machines with HTT**

On Windows, PBS calls the `CPUCount` Windows function, which reports whether hyper-threading is enabled. If hyper-threading is enabled, MoM uses the number of physical CPUs. If hyper-threading is not enabled, MoM uses the number of CPUs reported by the OS. MoM logs whether or not hyper-threading is enabled.

**10.6.3 Using Number of Physical CPUs**

If you do not wish to use hyper-threading, you can configure PBS to use the number of physical CPUs. Do this by setting `resources_available.ncpus` to the number of physical cpus:

```
Qmgr: set node <vnode name> resources_available.ncpus=<number of physical CPUs>
```

**10.6.4 Hyperthreading Caveats**

On a cpusetted system, NEVER change the value for `resources_available.ncpus`, `resources_available.vmem`, or `resources_available.mem`. 
Chapter 11

Managing Jobs

This chapter covers information on managing PBS jobs.

11.1 Routing Jobs

You can route jobs to various places and by various criteria. You can reject submission of jobs that request too much of a given resource. You can force jobs into the correct queues. You can have all jobs submitted to a routing queue, then route them to the correct execution queues. You can use peer scheduling to have jobs executed at other PBS complexes. You can use hooks to move jobs. For information on routing jobs, see section 4.8.39, “Routing Jobs”, on page 256.

11.2 Allocating Resources to Jobs

You can make sure that jobs request or inherit any resources required to manage those jobs. If a job does not request a resource, you can make sure that the resource is allocated to the job anyway.

In order for limits to be effective, each job must request each limited resource. For a complete description of how limits work, see section 5.15, “Managing Resource Usage”, on page 366.

You can create custom resources specifically to allocate them to jobs. These resources can be visible, alterable, and requestable by users, or invisible, unalterable, and unrequestable, or visible but unalterable and unrequestable. For instructions on creating invisible or unrequestable resources, see section 5.14.2.6, “Resource Permission Flags”, on page 328.
You can alter a job's resource request using the following methods:

- You can set defaults for resources at the server or at each queue. This way, you can have jobs inherit specific values for the resources by routing them to special queues, where they inherit the defaults. For how jobs inherit resources, see section 5.9.4, “Allocating Default Resources to Jobs”, on page 309. For how to specify default resources, see section 5.9.3, “Specifying Job Default Resources”, on page 305.

  For how resource defaults change when a job is moved, see section 5.9.4.3, “Moving Jobs Between Queues or Servers Changes Defaults”, on page 310.

- You can use a hook to assign a specific resource value to a job, if a job requests the wrong value for a resource. For how to use a hook to assign a resource to a job, see “Hooks” on page 413. For examples of using hooks to assign resources to jobs, see section 6.7.2, “Managing Job Resource Requests”, on page 426.

- You can use the `qalter` command to change a job's resource request. For how to use the qalter command, see “qalter” on page 127 of the PBS Professional Reference Guide.

- You can set default arguments the `qsub` command via the `default_qsub_arguments` server attribute. For how to use default arguments to `qsub`, see “Server Attributes” on page 316 of the PBS Professional Reference Guide.

### 11.2.1 Viewing Resources Allocated to a Job

#### 11.2.1.1 The exec vnode Attribute

The `exec vnode` attribute displayed via `qstat` shows the resources allocated from each vnode for the job.

The `exec vnode` line looks like:

```
exec_vnode = (<vnode name>:ncpus=W:mem=X)+(<vnode name>:ncpus=Y:mem=Z)
```

For example, a job requesting

```
-l select=2:ncpus=1:mem=1gb+1:ncpus=4:mem=2gb
```

gets an `exec vnode` of

```
exec_vnode = (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb) + (VNC:ncpus=4:mem=2gb)
```

Note that the vnodes and resources required to satisfy a chunk are grouped by parentheses. In the example above, if two vnodes on a single host were required to satisfy the last chunk, the `exec vnode` might be:

```
exec_vnode = (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb) + (VNC1:ncpus=2:mem=1gb)+VNC2:ncpus=2:mem=1gb)
```
Managing Jobs

Chapter 11

Note also that if a vnode is allocated to a job because the job requests an arrangement of exclhost, only the vnode name appears in the chunk. For example, if a job requesting

```
-1 select 2:ncpus=4 -l place = exclhost
```

is placed on a host with 4 vnodes, each with 4 CPUs, the exec_vnode attribute looks like this:

```
exec_vnode = (VN0:ncpus=4)+(VN1:ncpus=4)+(VN2)+(VN3)
```

### 11.2.1.2 The schedselect Attribute

The resources allocated from a vnode are only those specified in the job’s schedselect attribute. This job attribute is created internally by starting with the select specification and applying any server and queue default_chunk resource defaults that are missing from the select statement. The schedselect job attribute contains only vnode-level resources. The exec_vnode job attribute shows which resources are allocated from which vnodes. See “Job Attributes” on page 375 of the PBS Professional Reference Guide.

### 11.2.1.3 Resources for Requeued Jobs

When a job is requeued due to an error in the prologue or initialization, the job’s exec_host and exec_vnode attributes are cleared. The only exception is when the job is checkpointed and must be rerun on the exact same system. In this case, the exec_host and exec_vnode attributes are preserved.

### 11.3 Grouping Jobs By Project

#### 11.3.1 PBS Projects

In PBS, a project is a way to organize jobs independently of users and groups. A project is a tag that identifies a set of jobs. Each job’s project attribute specifies the job’s project. Each job can be a member of up to one project.

Projects are not tied to users or groups. One user or group may run jobs in more than one project. For example, user Bob runs JobA in ProjectA and JobB in ProjectB. User Bill runs JobC in ProjectA. User Tom runs JobD in ProjectB. Bob and Tom are in Group1, and Bill is in Group2.
11.3.2 Assigning Projects to Jobs

A job’s project can be set in the following ways:

- At submission, using the `qsub -P` option; see “qsub” on page 211 of the PBS Professional Reference Guide
- After submission, via the `qalter -P` option; see “qalter” on page 127 of the PBS Professional Reference Guide
- Via a hook; see Chapter 6, "Hooks", on page 413

11.3.3 Managing Resource Use by Project

PBS can apply limits to the amount of resources used by jobs in projects, or the number of running and queued jobs belonging to projects. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 367.

11.3.4 Managing Jobs by Project

You can arrange for the jobs belonging to a project to run on designated hardware; see section 4.3.4, “Allocating Resources by User, Project or Group”, on page 88. You can also run jobs belonging to a project in designated time slots; see section 4.3.6, “Scheduling Jobs into Time Slots”, on page 93. For more information on routing by project, see section 4.8.39, “Routing Jobs”, on page 256.

11.3.5 Viewing Project Information

Each job’s project, if any, is specified in its `project` attribute. To see the value of this attribute, use the `qstat -f` option. See “qstat” on page 196 of the PBS Professional Reference Guide.

11.3.6 Selecting Jobs by Project

You can select jobs according to their project using the `qselect -P` option. See “qselect” on page 185 of the PBS Professional Reference Guide.
11.3.7 Default Project Value

The default value for a job’s project attribute is “_.pbs_project_default”. Any job submitted without a specified value for the project attribute is given the default value. If you explicitly set the value to “_.pbs_project_default”, the server prints a warning message saying that the value has been set to the default. If you unset the value of the attribute in a hook, the value becomes the default value. Using `qalter -P ""` sets the value to the default.

11.3.8 Error Messages

When a job would exceed a limit by running, the job’s comment field is set to an error message. See “Run Limit Error Messages” on page 441 of the PBS Professional Reference Guide.

11.4 Job Prologue and Epilogue

PBS provides a facility for the administrator to run a site-supplied script or program before and/or after each job runs. This allows initialization or cleanup of resources, such as temporary directories or scratch files. The script or program that runs before the job is the prologue; the one that runs after the job is the epilogue.

The primary purpose of the prologue is to provide a site with some means of performing checks prior to starting a job. The epilogue can be used to requeue a checkpointed job. See section 9.3.7.3, “Requeueing via Epilogue”, on page 730.

Only one prologue and one epilogue may be used per PBS server. The same prologue and/or epilogue runs for every job in the complex.

Each script may be either a shell script or an executable object file.

11.4.1 When Prologue and Epilogue Run

The prologue runs before the job is executed. The epilogue runs after the job terminates for any reason, including normal termination, job deletion while running, error exit, or even if pbs_mom detects an error and cannot completely start the job. If the job is deleted while it is queued, then neither the prologue nor the epilogue is run.

If a prologue or epilogue script is not present, MoM continues in a normal manner.
11.4.2 Where Prologue and Epilogue Run

When multiple vnodes are allocated to a job, these scripts are run only by the MoM on the primary execution host.

The prologue runs with its current working directory set to PBS_HOME/mom_priv.

The epilogue runs with its current working directory set to the job's staging and execution directory. This is also where the job shell script is run.

11.4.3 Prologue and Epilogue Location

Both the prologue and the epilogue must reside in the PBS_HOME/mom_priv directory.

11.4.4 Prologue and Epilogue Requirements

In order to be run, the script must adhere to the following rules:

- The script must be in the PBS_HOME/mom_priv directory
- The prologue must have the exact name “prologue” under UNIX/Linux, or “prologue.bat” under Windows
- The epilogue must have the exact name “epilogue” under UNIX/Linux, or “epilogue.bat” under Windows
- The script must be written to exit with one of the zero or positive exit values listed in section 11.4.12, “Prologue and Epilogue Exit Codes”, on page 829. The negative values are set by MoM
- Under UNIX/Linux, the script must be owned by root, be readable and executable by root, and cannot be writable by anyone but root
- Under Windows, the script’s permissions must give “Full Access” to the local Administrators group on the local computer

11.4.5 Prologue and Epilogue Environment Variables

The prologue and epilogue run with the following set in their environment:

- The contents of the pbs_environment file
- The PBS_JOBDIR environment variable
11.4.6 Prologue and Epilogue Permissions

Both the prologue and epilogue are run under root on UNIX/Linux, or under an Admin-type account on Windows, and neither is included in the job session.

11.4.7 Prologue and Epilogue Limitations and Caveats

- The prologue cannot be used to modify the job environment or to change limits on the job.
- If any execjob_prologue hooks exist, they are run, and the prologue is not run.
- If any execjob_epilogue hooks exist, they are run, and the epilogue is not run.

11.4.8 Prologue and Epilogue Arguments

The prologue is called with the following arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv[1]</td>
<td>Job ID</td>
</tr>
<tr>
<td>argv[2]</td>
<td>User name under which the job executes</td>
</tr>
<tr>
<td>argv[3]</td>
<td>Group name under which the job executes</td>
</tr>
</tbody>
</table>

The epilogue is called with the following arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv[1]</td>
<td>Job ID</td>
</tr>
<tr>
<td>argv[2]</td>
<td>User name under which the job executes</td>
</tr>
<tr>
<td>argv[3]</td>
<td>Group name under which the job executes</td>
</tr>
<tr>
<td>argv[4]</td>
<td>Job name</td>
</tr>
<tr>
<td>argv[5]</td>
<td>Session ID</td>
</tr>
</tbody>
</table>
11.4.8.1 Epilogue Argument Caveats

Under Windows and with some UNIX shells, accessing `argv[10]` in the epilogue requires a shift in positional parameters. To do this, the script must do the following:

1. Call the arguments with indices 0 through 9
2. Perform a shift /8
3. Access the last argument using %9%

For example:

```bash
    cat epilogue
    > #!/bin/bash
    >
    > echo "argv[0] = $0" > /tmp/epiargs
    > echo "argv[1] = $1" >> /tmp/epiargs
    > echo "argv[2] = $2" >> /tmp/epiargs
    > echo "argv[3] = $3" >> /tmp/epiargs
    > echo "argv[4] = $4" >> /tmp/epiargs
    > echo "argv[5] = $5" >> /tmp/epiargs
    > echo "argv[6] = $6" >> /tmp/epiargs
    > echo "argv[7] = $7" >> /tmp/epiargs
    > echo "argv[8] = $8" >> /tmp/epiargs
    > echo "argv[9] = $9" >> /tmp/epiargs
    > shift
    > echo "argv[10] = $9" >> /tmp/epiargs
```
11.4.9 Standard Input to Prologue and Epilogue

Both scripts have standard input connected to a system-dependent file. The default for this file is /dev/null.

11.4.10 Standard Output and Error for Prologue and Epilogue

The standard output and standard error of the scripts are connected to the files which contain the standard output and error of the job. There is one exception: if a job is an interactive PBS job, the standard output and error of the epilogue is pointed to /dev/null because the pseudo-terminal connection used was released by the system when the job terminated.

11.4.11 Prologue and Epilogue Timeout

When the scheduler runs a job, it waits until the prologue has ended. To prevent an error condition within the prologue or epilogue from delaying PBS, MoM places an alarm around the script’s/program’s execution. The default value is 30 seconds. If the alarm timeout is reached before the script has terminated, MoM will kill the script. The alarm value can be changed via the $prologalarm MoM configuration parameter. See section 9.4.6, “Prologue & Epilogue Running Time”, on page 741.

11.4.12 Prologue and Epilogue Exit Codes

Normally, the prologue and epilogue programs should exit with a zero exit status. The prologue and epilogue should be written to exit with one of the zero or positive values listed here. When there is a problem with the script, MoM sets the exit value to one of the negative values. Exit status values and their impact on the job are listed in the following table:

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Meaning</th>
<th>Prologue</th>
<th>Epilogue</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>The script timed out (took too long).</td>
<td>The job will be requeued.</td>
<td>Ignored</td>
</tr>
<tr>
<td>-3</td>
<td>The wait(2) call waiting for the script to exit returned with an error.</td>
<td>The job will be requeued</td>
<td>Ignored</td>
</tr>
</tbody>
</table>
11.4.12.1 Logging Exit Status

MoM records in her log any case of a non-zero exit code, at event class 0x0001.

11.4.12.2 Epilogue Exit Status Caveats

Interactive-batch jobs cannot be requeued if the epilogue exits with a non-zero status. When this happens, these jobs are aborted.

11.4.13 Prologue Caveats

The administrator must exercise great caution in setting up the prologue to prevent jobs from being flushed from the system.

Table 11-3: Prologue and Epilogue Exit Codes

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Meaning</th>
<th>Prologue</th>
<th>Epilogue</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>The input file to be passed to the script could not be opened.</td>
<td>The job will be requeued.</td>
<td>Ignored</td>
</tr>
<tr>
<td>-1</td>
<td>The script has a permission error, is not owned by root, and/or is writable by others than root.</td>
<td>The job will be requeued.</td>
<td>Ignored</td>
</tr>
<tr>
<td>0</td>
<td>The script was successful.</td>
<td>The job will run.</td>
<td>Ignored</td>
</tr>
<tr>
<td>1</td>
<td>The script returned an exit value of 1.</td>
<td>The job will be aborted.</td>
<td>Ignored</td>
</tr>
<tr>
<td>&gt;1</td>
<td>The script returned a value greater than one.</td>
<td>The job will be requeued.</td>
<td>Ignored</td>
</tr>
<tr>
<td>2</td>
<td>The script returned a value of 2.</td>
<td>The job will be requeued.</td>
<td>If the job was checkpointed under the control of PBS, the job is requeued.</td>
</tr>
</tbody>
</table>
11.5 UNIX Shell Invocation

When PBS starts a job, it invokes the user’s login shell, unless the user submitted the job with the -S option. PBS passes the job script, which is a shell script, to the login process.

PBS passes the name of the job script to the shell program. This is equivalent to typing the script name as a command to an interactive shell. Since this is the only line passed to the script, standard input will be empty to any commands. This approach offers both advantages and disadvantages:

11.5.1 Advantages

- Any command which reads from standard input without redirection will get an EOF.
- The shell syntax can vary from script to script. It does not have to match the syntax for the user’s login shell. The first line of the script, even before any #PBS directives, should be

  `#!/shell`

  where `shell` is the full path to the shell of choice, `/bin/sh`, `/bin/csh`, ...

  The login shell will interpret the `#!/` line and invoke that shell to process the script.

11.5.2 Disadvantages

- An extra shell process is run to process the job script.
- If the script does start with a `#!/` line, the wrong shell may be used to interpret the script and thus produce errors.
- If a non-standard shell is used via the -S option, it will not receive the script, but its name, on its standard input.

11.6 Job’s Life Cycle

11.6.1 Sequence of Events for Start of Job

This is the order in which events take place on an execution host at the start of a job:

1. Licenses are obtained
2. Any job-specific staging and execution directories are created:
   - `PBS_JOBDIR` and job’s `jobdir` attribute are set to pathname of staging and execution
directory
- Files are staged in
3. $TMPDIR is created
4. The job’s cpusets are created
5. The prologue is executed
6. The job script is executed

11.6.2 Sequence of Events for End of Job

This is the order in which events generally take place at the end of a job:
7. The job script finishes
8. The epilogue is run
9. The obit is sent to the server
10. Any specified file staging out takes place, including stdout and stderr
11. Files staged in or out are removed
12. Any job-specific staging and execution directories are removed
13. Job files are deleted
14. Application licenses are returned to pool
15. The job’s cpusets are destroyed

11.7 When Job Attributes are Set

The attributes of a job are set at various points in the life of the job. For a description of each job attribute, see “Job Attributes” on page 375 of the PBS Professional Reference Guide.
11.7.1 Job Attributes Set By qsub Command

Before the job is passed to the server, the qsub command sets these job attributes, in this order:

1. Attributes specified as options on the command line
2. Attributes specified in #PBS directives within the job script
3. Job attributes specified in the default_qsub_arguments server attribute
4. If the following job attributes have not already been set, they are set as follows:
   - Job_Name: set to the file name of the job script, or to "STDIN" if the script is entered via standard input
   - Checkpoint: set to "u" for unspecified.
   - Hold_Types: set to "n"
   - Join_Path: set to "n"
   - Keep_Files: set to "n"
   - Mail_Points: set to "a" for abort
   - Priority: set to 0 (zero)
   - Rerunnable: set to True
   - run_count: can be set by job submitter
   - Variable_List: the qsub command sets the following variables and appends them to the existing value of Variable_List: PBS_0_HOME, PBS_0_LANG, PBS_0_LOGNAME, PBS_0_PATH, PBS_0_MAIL, PBS_0_SHELL, PBS_0_WORKDIR, PBS_0_TZ, and PBS_O_SYSTEM
   - Submit_arguments: set to any submission arguments on the command line

11.7.2 Job Attributes Set at Server

When the job is passed from the qsub command to the server, the raw job information is available to any job submission hooks, which can alter the information. Once the job is at the server, the server sets the following attributes:

- JobOwner: set to <username>@<submission host name>
- Variable_List: the following are added to the job’s Variable_List attribute:
Chapter 11

Managing Jobs

PBS_O_QUEUE, PBS_O_HOST

• Output_Path: if not yet specified, the Output_Path attribute is set
• Error_Path: if not yet specified, the Error_Path attribute is set
• Rerunable: if the job is interactive, the Rerunable attribute is set to False
• run_count: incremented each time job is run
• project: if unset, the project attribute is set to "_pbs_project_default".
• Read-only attributes: the server sets the job’s read-only attributes; see "The following job attributes are read-only:" on page 389
• Resource_List: adjusted to include inherited resources specified in the queue and server Resources_Default attributes, if those resources are not yet in the list

11.7.3 Attributes Changed by Operations on Jobs

11.7.3.1 Attributes Changed When Moving Job

If you move a job to a different queue or server, any default resources from the current queue or server are removed, and new defaults are inherited. See section 5.9.4.3, "Moving Jobs Between Queues or Servers Changes Defaults", on page 310. For information on the qmove command, see “qmove” on page 172 of the PBS Professional Reference Guide.

11.7.3.2 Attributes Changed When Altering Job

When the qalter command is used to alter a job, the changes to the job are changes to the equivalent job attributes. See “qalter” on page 127 of the PBS Professional Reference Guide.

11.7.3.3 Attributes Changed When Requeueing or Rerunning a Job

When a job is requeued or rerun, its exec_vnode and/or exec_host attributes may be changed. The job may end up running on different vnodes. See “qrerun” on page 177 of the PBS Professional Reference Guide.

Each time a job is run, its run_count attribute is incremented by the server.
11.7.3.4 Attributes Changed by Holding or Releasing a Job

When a job is held using the `qhold` command, or released using the `qrls` command:

- The `Hold_Types` attribute reflects the change
- The `job_state` attribute may be changed


11.7.3.5 Attributes Changed by Suspending or Resuming a Job

When a job is suspended or resumed using the `qsig` command, the job’s `job_state` attribute reflects the change in state. See “qsig” on page 192 of the PBS Professional Reference Guide.

11.8 Job Termination

11.8.1 Normal Job Termination

When there is no $action terminate script and a running job is terminated, via the `qdel <job ID>` command, because of a server shutdown, or because the job has exceeded a limit, PBS waits for a configurable amount of time between sending a SIGTERM and a SIGKILL signal to the job. The amount of time is specified in the `kill_delay` queue attribute. The default value for this attribute is 10 seconds. PBS takes the following steps.

For a single-vnode job:

1. PBS sends the job a SIGTERM
2. PBS waits for the amount of time specified in the `kill_delay` queue attribute
3. PBS sends the job a SIGKILL

For a multi-vnode job:

1. Mother superior sends a SIGTERM to all processes on the primary execution host
2. If any of the processes of the top task of the job are still running, PBS waits a minimum of `kill_delay` seconds
3. Mother Superior sends a SIGKILL to all remaining job processes on the primary execution host
4. The subordinate MoMs send a SIGKILL to all their processes belonging to this job
11.8.2 Using the qdel Command to Terminate a Job

You can delete a job using the qdel command. See “qdel” on page 142 of the PBS Professional Reference Guide.

qdel <job ID>

If there is an $action terminate script, it is used to terminate the job.

If there is no $action terminate script, the SIGTERM-delay-SIGKILL sequence described in section 11.8.1, “Normal Job Termination”, on page 835 is used to terminate the job.

This command does not terminate provisioning jobs.

qdel -Wforce <job ID>

If MoM is reachable, MoM sends the job a SIGKILL signal, and files are staged out. If MoM is unreachable, the server discards the job. The job may or may not continue to run on the execution host(s).

This command terminates provisioning jobs.

11.8.3 Killing Job Processes

If you need to kill job processes, you can use the printjob command to find the job’s session ID, and then kill those processes. See “printjob” on page 125 of the PBS Professional Reference Guide.

11.8.4 Configuring Site-specific Job Termination

The default behavior of PBS is for MoM to terminate a job under the following circumstances:

- The job's usage of a resource exceeds the limit requested
- The job is deleted by the Server on shutdown
- The job is deleted via the qdel command

MoM normally uses SIGTERM, waits for the amount of time specified in the server’s kill_delay attribute, then issues a SIGKILL. See section 11.8, “Job Termination”, on page 835.

You may want PBS to run your own job termination script in place of the normal action. The termination script is run in place of a SIGTERM. The termination script runs only on the primary execution host. After the top job process is terminated, a KILL signal is sent to any other job processes running on other hosts.
You can define the desired termination behavior by specifying the script you want to run in the \texttt{\$action terminate} parameter in the Version 1 configuration file. The \texttt{\$action terminate} parameter takes this form:

\texttt{\$action terminate <timeout> ! <path to script> [args]}  

Where

\texttt{<timeout>} is the time, in seconds, allowed for the script to complete. A value of \texttt{zero} (0) indicates infinite time is allowed for the script to run.

\texttt{<path to script>} is the path to the script. If it is a relative path, it is evaluated relative to the \texttt{PBS\_HOME/mom\_priv} directory.

\texttt{<args>} are optional arguments to the script. Values for \texttt{<args>} may be any string not starting with a percent sign ("%")

Arguments with a percent sign, making up any of the following keywords, are replaced by MoM with the corresponding value:

\textbf{Table 11-4: \texttt{\$action terminate} Keywords}

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value Used by MoM</th>
</tr>
</thead>
<tbody>
<tr>
<td>%jobid</td>
<td>Job ID</td>
</tr>
<tr>
<td>%sid</td>
<td>Session ID of task (job)</td>
</tr>
<tr>
<td>%uid</td>
<td>Execution UID of job</td>
</tr>
<tr>
<td>%gid</td>
<td>Execution GID of job</td>
</tr>
<tr>
<td>%login</td>
<td>Login name associated with UID</td>
</tr>
<tr>
<td>%owner</td>
<td>Job owner in form name@host</td>
</tr>
<tr>
<td>%auxid</td>
<td>Auxiliary ID (system-dependent)</td>
</tr>
</tbody>
</table>

\textbf{11.8.4.1 Requirements for Termination Script}

The script should exit with a value of \texttt{zero} when the job is terminated successfully. If the script exits successfully (with a zero exit status and before the time-out period), PBS does not send any signals or attempt to terminate the job. It is the responsibility of the termination script in this situation to ensure that the job has been terminated.

The script should exit with a non-zero value if the job was not successfully terminated. If the script exits with a non-zero exit status, the job is sent \texttt{SIGKILL} by PBS.
If the script does not complete in the time-out period, it is aborted and the job is sent SIGKILL.

### 11.8.4.2 Examples of Configuring Termination

**UNIX:**

Example 11-1: To use a 60-second timeout, run `PBS_HOME/mom_priv/endjob.sh`, and pass the job’s session ID, user ID, and PBS jobs ID to the script:

```bash
$action terminate 60 !endjob.sh %sid %uid %jobid
```

Example 11-2: To use an infinite timeout, run the system `kill` command with the signal 13, and pass the job’s session ID:

```bash
$action terminate 0 !/bin/kill -13 %sid
```

**Windows:**

Example 11-3: To use a 60-second timeout, run `endjob.bat`, and pass the job’s session ID, user ID, and PBS jobs ID to the script:

```bash
$action terminate 60 !endjob.bat %sid %uid %jobid
```

Example 11-4: To use an infinite timeout, run the `pbskill` command, and pass the job’s session ID:

```bash
$action terminate 0 !"C:/Program Files/PBS Pro/exec/bin/pbskill" %sid
```

### 11.8.4.3 Caveats and Restrictions on Termination

Under Windows, `<path to script>` must have a “.bat” suffix since it will be executed under the Windows command prompt `cmd.exe`. If the `<path to script>` specifies a full path, be sure to include the drive letter so that PBS can locate the file. For example, `C:\winnt\temp\terminate.bat`. The script must be writable by no one but an Administrator-type account.
11.9 Job Exit Codes

The exit value of a job may fall in one of three ranges, listed in the following table:

**Table 11-5: Job Exit Code Ranges**

<table>
<thead>
<tr>
<th>Exit Code Range</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X &lt; 0$</td>
<td>The job could not be executed</td>
<td>See section 11.9.1, “Negative Job Exit Codes”, on page 839</td>
</tr>
<tr>
<td>$0 \leq X &lt; 128$</td>
<td>Exit value of shell</td>
<td>See section 11.9.2, “Job Exit Codes Between 0 and 128 (or 256)”, on page 841</td>
</tr>
<tr>
<td>$X \geq 128$</td>
<td>Job was killed with a signal</td>
<td>See section 11.9.3, “Job Exit Codes $\geq 128$ (or 256)”, on page 841</td>
</tr>
</tbody>
</table>

11.9.1 Negative Job Exit Codes

This is a PBS special return value indicating that the job could not be executed. These negative values are listed in the table below:

**Table 11-6: Job Exit Codes**

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>JOB_EXEC_OK</td>
<td>Job execution was successful</td>
</tr>
<tr>
<td>-1</td>
<td>JOB_EXEC_FAIL1</td>
<td>Job execution failed, before files, no retry</td>
</tr>
<tr>
<td>-2</td>
<td>JOB_EXEC_FAIL2</td>
<td>Job execution failed, after files, no retry</td>
</tr>
<tr>
<td>-3</td>
<td>JOB_EXEC_RETRY</td>
<td>Job execution failed, do retry</td>
</tr>
<tr>
<td>-4</td>
<td>JOB_EXEC_INITABT</td>
<td>Job aborted on MoM initialization</td>
</tr>
</tbody>
</table>
### Table 11-6: Job Exit Codes

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5</td>
<td>JOB_EXEC_INITRST</td>
<td>Job aborted on MoM initialization, checkpoint, no migrate</td>
</tr>
<tr>
<td>-6</td>
<td>JOB_EXEC_INITRMG</td>
<td>Job aborted on MoM initialization, checkpoint, ok migrate</td>
</tr>
<tr>
<td>-7</td>
<td>JOB_EXEC_BADRESRT</td>
<td>Job restart failed</td>
</tr>
<tr>
<td>-8</td>
<td>JOB_EXEC_GLOBUS_INIT_RETRY</td>
<td>Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No longer used. Initialization of Globus job failed; do retry</td>
</tr>
<tr>
<td>-9</td>
<td>JOB_EXEC_GLOBUS_INIT_FAIL</td>
<td>Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No longer used. Initialization of Globus job failed; no retry</td>
</tr>
<tr>
<td>-10</td>
<td>JOB_EXEC_FAILUID</td>
<td>Invalid UID/GID for job</td>
</tr>
<tr>
<td>-11</td>
<td>JOB_EXEC_RERUN</td>
<td>Job was rerun</td>
</tr>
<tr>
<td>-12</td>
<td>JOB_EXEC_CHKPK</td>
<td>Job was checkpointed and killed</td>
</tr>
<tr>
<td>-13</td>
<td>JOB_EXEC_FAIL_PASSWORD</td>
<td>Job failed due to a bad password</td>
</tr>
<tr>
<td>-14</td>
<td>JOB_EXEC_RERUN_ON_SIS_FAIL</td>
<td>Job was requeued (if runnable) or deleted (if not) due to a communication failure between Mother Superior and a Sister</td>
</tr>
<tr>
<td>-15</td>
<td>JOB_EXEC_QUERST</td>
<td>Requeue job for restart from checkpoint</td>
</tr>
</tbody>
</table>
11.9.2 Job Exit Codes Between 0 and 128 (or 256)

This is the exit value of the top process in the job, typically the shell. This may be the exit value of the last command executed in the shell or the .logout script if the user has such a script (csh).

11.9.3 Job Exit Codes >= 128 (or 256)

This means the job was killed with a signal. The signal is given by X modulo 128 (or 256). For example an exit value of 137 means the job's top process was killed with signal 9 (137 % 128 = 9).

The exit status values greater than 128 (or 256) indicate which signal killed the job. Depending on the system, values greater than 128 (or on some systems 256; see wait(2) or wait-pid(2) for more information), are the value of the signal that killed the job.

To interpret (or “decode”) the signal contained in the exit status value, subtract the base value from the exit status. For example, if a job had an exit status of 143, that indicates the job was killed via a SIGTERM (e.g. 143 - 128 = 15, signal 15 is SIGTERM). See the kill(1) manual page for a mapping of signal numbers to signal name on your operating system.

---

### Table 11-6: Job Exit Codes

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-16</td>
<td>JOB_EXECEXECFAILHOOK_RERUN</td>
<td>Job execution failed due to hook rejection; requeue for later retry</td>
</tr>
<tr>
<td>-17</td>
<td>JOB_EXECEXECFAILHOOK_DELETE</td>
<td>Job execution failed due to hook rejection; delete the job at end</td>
</tr>
<tr>
<td>-18</td>
<td>JOB_EXECEXEXEC_HOOK_RERUN</td>
<td>A hook requested for job to be requeued</td>
</tr>
<tr>
<td>-19</td>
<td>JOB_EXECEXEXEC_HOOK_DELETE</td>
<td>A hook requested for job to be deleted</td>
</tr>
<tr>
<td>-20</td>
<td>JOB_EXECEXEXEC_RERUN_MS_FAIL</td>
<td>Mother superior connection failed</td>
</tr>
</tbody>
</table>

---
11.9.4 Logging Job Exit Codes

The exit status of jobs is recorded in the PBS Server logs and the accounting logs.

11.10 Rerunning or Requeuing a Job

You can re-run a job using the `qrerun` command. To re-run a job means to kill it, and requeue it in the execution queue from which it was run. See “qrerun” on page 177 of the PBS Professional Reference Guide.

11.10.1 Output from a Re-run Job

When you re-run a job, the job’s existing standard output and error files are copied back to the server host and stored in `PBS_HOME/spool`. They are then sent with the job to MoM when the job is again run. The output of a job that is re-run is appended to the output from prior runs of the same job.

11.10.2 Caveats for `qrerun`

- Jobs lose their queue wait time when they are requeued, including when they are check-pointed or requeued during preemption.

11.10.3 Requeueing Caveats

- When requeueing a job fails, for example because the queue does not exist, the job is deleted.
- If a job’s `run_count` attribute is already at the limit (20), and you requeue the job, the job will be held the next time the scheduler tries to run it.

11.10.4 Caveats for Jobs Started by PBS

PBS attempts to run a job a certain number of times before placing a hold on the job. You cannot prevent a job from being held after this number of attempts. You must explicitly release the hold.
11.11 Job IDs

11.11.1 Format of Job IDs

Job Identifier

\[ \text{sequence_number}\.\text{server_name}@\text{server} \]

Job Array Identifier

Job array identifiers are a sequence number followed by square brackets:

\[ \text{sequence_number}[\text{server_name}@\text{server}] \]

Example:

1234[

Note that some shells require that you enclose a job array ID in double quotes.

11.11.2 Range of IDs

The largest possible job ID is the 7-digit number \(9999999\). After this has been reached, job IDs start again at zero.

11.11.3 Job IDs and Moving Jobs

If a job is moved from one server to another, the job’s ID does not change.

11.11.4 Job IDs and Requeueing and Checkpoint/Restart

If a job is requeued without being checkpointed, or checkpointed and requeued, it keeps its original job ID.

11.12 Where to Find Job Information

Information about jobs is found in \(\text{PBS\_HOME/server\_priv/jobs}\) and \(\text{PBS\_HOME/mom\_priv/jobs}\).

11.12.1 Deleted Jobs

If PBS tries to requeue a job and cannot, for example when the queue doesn’t exist, the job is deleted.
11.12.2 Finding the cpuset Assigned to a Job

To find out which cpuset is assigned to a running job, the alt_id job attribute has a field called cpuset that will show this information. The cpusets are created with the name of the jobid for which they are created.

11.12.3 Failed Jobs

Once a job has experienced a certain number of failures, PBS holds the job.

11.12.4 Job Information When Server is Down

When the PBS server is down, you can use the pbs_dataservice command to start the PBS data service by hand, and then run the printjob command at the server host. See “pbs_dataservice” on page 46 of the PBS Professional Reference Guide and “printjob” on page 125 of the PBS Professional Reference Guide.

11.12.5 Job Information on Execution Host

You can use the printjob command to look at job information on the execution host. See “printjob” on page 125 of the PBS Professional Reference Guide.

11.13 Job Directories

PBS jobs use two kinds of directories. The first is the job’s staging and execution directory. Files are staged into and out of this directory, and this is the directory where the job script executes.

The second is the job’s temporary directory, where the job can create scratch files if necessary. The root of this directory is specified in the $tmpdir MoM configuration parameter. PBS creates the temporary directory, then sets the TMPDIR job environment variable to the path of the temporary directory. The job can then use this environment variable. See section 11.14.1, “Creation of TMPDIR”, on page 849.

11.13.1 The Job’s Staging and Execution Directories

A job’s staging and execution directory is the directory to which input files are staged, and from which output files are staged. It is also the current working directory for the job script, for tasks started via the pbs_tm() API, and for the epilogue.
Managing Jobs

Each PBS user may submit several jobs at once. Each job may need to have data files staged in or out. Each execution host needs a staging and execution directory for jobs. PBS can provide a job-specific staging and execution directory on each execution host for each job. The job’s sandbox attribute controls whether PBS creates a staging and execution directory for each job, or uses the user’s home directory for staging and execution.

PBS stages files to and from the primary execution host only. If the execution hosts use a shared file system, the staging and execution directory is available everywhere the job needs it.

When a job uses a job-specific staging and execution directory created by PBS, PBS does not require the job’s owner to have a home directory on the execution host(s), as long as each MoM’s $jobdir_root configuration option is set, and is set to something other than the user’s home directory.

Staging is specified via the job’s stagein and stageout attributes. The format is the following:

```
execution_path@[storage_host:]storage_path
```

The execution_path is the path to the staging and execution directory. On stagein, storage_path is the path where the input files normally reside, and on stageout, storage_path is the path where output files will end up.

### 11.13.1.1 The Job’s sandbox Attribute

If the job’s sandbox attribute is set to PRIVATE, PBS creates a job-specific staging and execution directory for that job. If sandbox is unset, or is set to HOME, PBS uses the user’s home directory as the job’s staging and execution directory. Using the server’s default_qsub_arguments attribute, you can specify the default for the sandbox attribute for all jobs. By default, the sandbox attribute is not set.

The user can set the sandbox attribute via qsub, for example:

```
qsub -Wsandbox=PRIVATE
```

The -Wsandbox option to qsub overrides default_qsub_arguments. The job’s sandbox attribute cannot be altered while the job is executing.

<table>
<thead>
<tr>
<th>Job’s sandbox attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>not set</td>
<td>Job’s staging and execution directory is the user’s home directory</td>
</tr>
</tbody>
</table>
11.13.1.2 Options, Attributes and Environment Variables Affecting Staging

The environment variable PBS_JOBDIR is set to the pathname of the staging and execution directory on the primary execution host. PBS_JOBDIR is added to the job script process, any job tasks created by the pbs_t_m() API, the prologue and epilogue, and the MoM $action scripts.

The job’s jobdir attribute is read-only, and is also set to the pathname of the staging and execution directory on the primary execution host. The jobdir attribute can be viewed using the -f option to qstat.

The following table lists the options, attributes, etc., affecting staging:

### Table 11-8: Options, Attributes, Environment Variables, etc., Affecting Staging

<table>
<thead>
<tr>
<th>Option, Attribute, Environment Variable, etc.</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoM’s $jobdir_root option</td>
<td>Directory under which PBS creates job-specific staging and execution directories. Defaults to user’s home directory if unset. If $jobdir_root is unset, the user’s home directory must exist. If $jobdir_root does not exist when MoM starts, MoM will abort. If $jobdir_root does not exist when MoM tries to run a job, MoM will kill the job. Permissions on the directory specified in this option must be 1777.</td>
</tr>
</tbody>
</table>
### Table 11-8: Options, Attributes, Environment Variables, etc., Affecting Staging

<table>
<thead>
<tr>
<th>Option, Attribute, Environment Variable, etc.</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoM’s $usecp option</td>
<td>Tells MoM where to look for files in a shared file system; also tells MoM that she can use the local copy agent for these files.</td>
</tr>
<tr>
<td>Job’s sandbox attribute</td>
<td>Determines which directory PBS uses for the job’s staging and execution. If value is <em>PRIVATE</em>, PBS uses a job-specific directory it creates under the location specified in the MoM $jobdir_root configuration option. If value is <em>HOME</em> or is unset, PBS uses the user’s home directory for staging and execution. User-settable per-job via <code>qsub -W</code> or through a PBS directive. See the <code>pbs_mom.8B</code> man page.</td>
</tr>
<tr>
<td>Job’s stagein attribute</td>
<td>Sets list of files or directories to be staged in. User-settable per job via <code>qsub -W</code>.</td>
</tr>
<tr>
<td>Job’s stageout attribute</td>
<td>Sets list of files or directories to be staged out. User-settable per job via <code>qsub -W</code>.</td>
</tr>
<tr>
<td>Job’s jobdir attribute</td>
<td>Set to pathname of staging and execution directory on primary execution host. Read-only; viewable via <code>qstat -f</code>.</td>
</tr>
<tr>
<td>Job’s Keep_Files attribute</td>
<td>Determines whether output and/or error files remain on execution host. User-settable per job via <code>qsub -k</code> or through a PBS directive. If the <em>Keep_Files</em> attribute is set to <em>o</em> and/or <em>e</em> (output and/or error files remain in the staging and execution directory) and the job’s <em>sandbox</em> attribute is set to <em>PRIVATE</em>, standard out and/or error files are removed when the staging directory is removed at job end along with its contents.</td>
</tr>
<tr>
<td>Job’s PBS_JOBDIR environment variable</td>
<td>Set to pathname of staging and execution directory on primary execution host. Added to environments of job script process, <code>pbs_tm</code> job tasks, prologue and epilogue, and MoM $action scripts.</td>
</tr>
</tbody>
</table>
Managing Jobs

Table 11-8: Options, Attributes, Environment Variables, etc., Affecting Staging

<table>
<thead>
<tr>
<th>Option, Attribute, Environment Variable, etc.</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job’s TMPDIR environment variable</td>
<td>Location of job-specific scratch directory.</td>
</tr>
<tr>
<td>PBS_RCP string in pbs.conf</td>
<td>Location of rcp command</td>
</tr>
<tr>
<td>PBS_SCP string in pbs.conf</td>
<td>Location of scp command; setting this parameter causes PBS to first try scp rather than rcp for file transport.</td>
</tr>
<tr>
<td>Server’s default_qsub_arguments attribute</td>
<td>Can contain a default for job’s sandbox (and other) attributes.</td>
</tr>
</tbody>
</table>

11.13.1.3 Getting Information About the Job’s Staging and Execution Directory

The job’s jobdir attribute is viewable via qstat or the equivalent API while a job is executing. The value of jobdir is not retained if a job is rerun; it is undefined whether jobdir is visible or not when the job is not executing.

11.13.1.4 Example of Setting Location for Creation of Staging and Execution Directories

To make it so that jobs with sandbox=PRIVATE have their staging and execution directories created under /scratch, as /scratch/<job-specific_dir_name>, put the following line in MoM’s configuration file:

```
$jobdir_root /scratch
```

11.13.1.5 Staging and Execution Directory Caveats

If the user home directory is NFS mounted, and you want to use sandbox=PRIVATE, then root must be allowed write privilege on the NFS filesystem on which the users’ home directories reside.

The directory specified in MoM’s $jobdir_root parameter must have permissions set to 1777.
11.14 The Job’s Lifecycle

11.14.1 Creation of TMPDIR

For each host allocated to the job, PBS creates a job-specific temporary scratch directory for this job. The root of TMPDIR is set by MoM to the value of MoM’s $tmpdir configuration option. PBS sets TMPDIR to the pathname of the job-specific temporary scratch directory. This directory is for the use of the job, not PBS. This directory and its contents are removed when the job is finished.

The recommended TMPDIR configuration is to have a separate, local directory on each host. If the temporary scratch directory cannot be created, the job is killed.

11.14.2 Choice of Staging and Execution Directories

If the job’s sandbox attribute is set to PRIVATE, PBS creates job-specific staging and execution directories for the job. If the job’s sandbox attribute is set to HOME, or is unset, PBS uses the user’s home directory for staging and execution. The staging and execution directory may be shared (e.g., cross-mounted) among all the hosts allocated to the job, or each host may use a separate directory. This is true whether or not the directory is the user’s home directory.

11.14.2.1 Choosing Job-specific Staging and Execution Directories

When PBS creates a job-specific staging and execution directory, it does so under the directory specified in the MoM configuration option $jobdir_root. If the $jobdir_root option is not set, job-specific staging and execution directories are created under the user’s home directory.

If the staging and execution directory is accessible on all of the job’s execution hosts, these hosts will log the following message at the 0x0400 event class:

“the staging and execution directory <full path> already exists”.

If the staging and execution directory is not cross-mounted so that it is accessible on all the job's execution hosts, each secondary host also creates a directory using the same base name as was used on the primary host.

If the staging and execution directory cannot be created the job is aborted. The following error message is logged at 0x0001 event class:

“unable to create the job directory <full path>“.

When PBS creates a directory, the following message is logged at 0x0008 event class:

“created the job directory <full path>”
11.14.2.1.i  Job-specific Staging and Execution Directory Caveats

- You should not depend on any particular naming scheme for the new directories that PBS creates for staging and execution. The pathname to each directory on each node may be different, since each depends on the corresponding MoM’s $jobdir_root.
- The permissions for the directory specified in $jobdir_root must be 1777.

11.14.2.2  Choosing User’s Home Directory as Staging and Execution Directory

If the job’s sandbox attribute is unset or is set to HOME, PBS uses the user’s home directory for the job’s staging and execution directory.

The user must have a home directory on each execution host. The absence of the user's home directory is an error and causes the job to be aborted.

11.14.3  Setting PBS_JOBDIR and the Job’s jobdir Attribute

PBS sets PBS_JOBDIR and the job’s jobdir attribute to the pathname of the staging and execution directory.

11.14.4  Staging Files Into Staging and Execution Directories

PBS evaluates execution_path and storage_path relative to the staging and execution directory given in PBS_JOBDIR, whether this directory is the user’s home directory or a job-specific directory created by PBS. PBS stages files to the primary execution host only. Staging is done as the job owner.

PBS uses local file transfer mechanisms where possible. For remote file transfers, PBS uses the mechanism you specify. See section 12.9, “Setting File Transfer Mechanism”, on page 881.

11.14.5  Running the Prologue

The MoM’s prologue is run on the primary host as root, with the current working directory set to PBS_HOME/mom_priv and with PBS_JOBDIR set in its environment.
11.14.6  Job Execution

PBS runs the job script on the primary host as the user. PBS also runs any tasks created by the job via the pbs_tm() API as the user. The job script and tasks are executed with their current working directory set to the job's staging and execution directory, and with PBS_JOBDIR and TMPDIR set in their environment. The job attribute jobdir is set to the pathname of the staging and execution directory on the primary host.

11.14.7  Standard Out, Standard Error and TMPDIRs

The job's stdout and stderr files are created directly in the job's staging and execution directory on the primary execution host.

11.14.7.1  Output and Error with Job-specific Staging and Execution Directories

If the qsub -k option is used, the stdout and stderr files will not be automatically copied out of the staging and execution directory at job end; they will be deleted when the directory is automatically removed.

11.14.7.2  Output and Error with User’s Home Directory as Staging and Execution Directory

If the -k option to qsub is used, standard out and/or standard error files are retained on the primary execution host instead of being returned to the submission host, and are not deleted after job end.

11.14.8  Running the Epilogue

PBS runs MoM's epilogue script on the primary host as root. The epilogue is executed with its current working directory set to the job's staging and execution directory, and with PBS_JOBDIR set in its environment.

11.14.9  Staging Files Out and Removing Execution Directory

When PBS stages files out, it evaluates execution_path and storage_path relative to PBS_JOBDIR. Files that cannot be staged out are saved in PBS_HOME/undelivered. PBS stages files out from the primary execution host only. Staging is done as the job owner.
PBS uses local file transfer mechanisms where possible. For remote file transfers, PBS uses the mechanism you specify. See section 12.9, “Setting File Transfer Mechanism”, on page 881.

When the job is done, PBS writes the final job accounting record and purges job information from the Server’s database.

11.14.9.1 Staging Out with Job-specific Staging and Execution Directories

If PBS created job-specific staging and execution directories for the job, it cleans up at the end of the job. If no errors are encountered during stageout and all stageouts are successful, the staging and execution directory and all of its contents are removed, on all execution hosts.

Files to be staged out are deleted all together, only after successful stageout of all files. If any errors are encountered during stageout, no files are deleted on the primary execution host, and the execution directory is not removed.

If PBS created job-specific staging and execution directories on secondary execution hosts, those directories and their contents are removed at the end of the job, regardless of stageout errors.


Files that are successfully staged out are deleted immediately, without regard to files that were not successfully staged out.

11.14.10 Removing TMPDIRs

PBS removes all TMPDIRs, along with their contents.

11.15 Managing Job History

11.15.1 Introduction

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used.
PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server.

11.15.2 Definitions

Moved jobs
Jobs which were moved to another server

Finished jobs
Jobs whose execution is done, for any reason:
- Jobs which finished execution successfully and exited
- Jobs terminated by PBS while running
- Jobs whose execution failed because of system or network failure
- Jobs which were deleted before they could start execution

History jobs
Jobs which will no longer execute at this server:
- Moved jobs
- Finished jobs

11.15.3 Job History Information

PBS can keep all job attribute information, including the following kinds of job history information:
- Submission parameters
- Whether the job started execution
- Whether execution succeeded
- Whether staging out of results succeeded
- Which resources were used

PBS keeps job history for the following jobs:
- Jobs that have finished execution
- Jobs that were deleted
- Jobs that were moved to another server

You can configure whether PBS preserves job history, and for how long. The job history for finished and moved jobs is preserved and available for the specified duration. After the duration has expired, PBS deletes the job history information and it is no longer available.
11.15.4 Configuring Job History Management

To configure job history, you enable it and you set the job history duration. You configure PBS to manage job history using the following server attributes:

```
job_history_enable
   Enables or disables job history management. Setting this attribute to True enables job history management.
   Format: Boolean.
   Default: False

job_history_duration
   Specifies the length of time that PBS will keep each job's history.
   Format: duration: [hours:]minutes:[seconds][.milliseconds]
   Default: Two weeks (336:00:00)
```

11.15.4.1 Enabling Job History

To enable job history management, set the server's `job_history_enable` attribute to `True`:

```
Qmgr: set server job_history_enable=True
```

11.15.4.2 Setting Job History Duration

To set the length of time that job history is preserved, set the server's `job_history_duration` attribute to the desired duration:

```
Qmgr: set server job_history_duration=<duration>
```

If the job history duration is set to zero, no history is preserved.

If job history is enabled and job history duration is unset, job history information is kept for the default 2 weeks.

11.15.5 Changing Job History Settings

11.15.5.1 Disabling Job History

If job history is being preserved, and you unset the `job_history_enable` server attribute, PBS deletes all job history information. This information is no longer available.
11.15.5.2 Enabling Job History

If job history is not being preserved, and you set the `job_history_enable` server attribute, PBS begins preserving job history information for any jobs that are queued or running.

11.15.5.3 Modifying Job History Duration

Every job’s history duration is set to the current value of the `job_history_duration` server attribute.

Example 11-5: Reducing job history duration:

The value of `job_history_duration` was “00:10:00” when a job finished execution. After 2 minutes, you change the duration to “00:06:00”. This job’s history is kept for a total of 6 minutes.

Example 11-6: Increasing job history duration:

The value of `job_history_duration` was “00:10:00” when a job finished execution. After 8 minutes you change the duration to “00:30:00”. This job’s history is kept for a total of 30 minutes.

Example 11-7: Increasing job history duration:

The value of `job_history_duration` was “00:10:00” when a job finished execution. After 11 minutes you change the duration to “00:30:00”. This job’s history is kept for a total of 10 minutes. The job’s history is deleted after it is kept for 10 minutes.

11.15.6 Backward Compatibility

To have PBS behave as it did before the job history management feature was introduced, disable job history management. Do one of the following:

- Set the server’s `job_history_enable` attribute to `False`:
  
  Qmgr: set server job_history_enable=False

- Unset the server’s `job_history_enable` attribute:

  Qmgr: unset server job_history_enable

- Set the value of the server’s `job_history_duration` attribute to zero, by doing one of the following:

  Qmgr: set server job_history_duration=0
  Qmgr: set server job_history_duration=00:00
  Qmgr: set server job_history_duration=00:00:00
11.15.7 Logging Moved Jobs

Jobs can be moved to another server for one of the following reasons:

- Moved for peer scheduling
- Moved via the qmove command
- Job was submitted to a routing queue, then routed to a destination queue at another server

When a job is moved, the server logs the event in the server log and the accounting log. The server log messages are logged at log event class 0x0008.

Format for the server log file:

```
7/08/2008 16:17:38;0008;Server@serverhost1;Job;
97.serverhost1.domain.com;Job moved to destination: workq@serverhost2
```

Format for the accounting log entry:

```
7/08/2008
16:17:38;M;97.serverhost1.domain.com;destination=workq@serverhost2
```

Record type: **M** (moved job)

11.15.8 Deleting Moved Jobs and Job Histories

You can use the **qdel -x** option to delete job histories. This option also deletes any specified jobs that are queued, running, held, suspended, finished, or moved. When you use this, you are deleting the job and its history in one step. If you use the qdel command without the -x option, you delete the job, but not the job history, and you cannot delete a moved or finished job. See "qdel" on page 142 of the PBS Professional Reference Guide.

11.15.9 Job History Caveats

- Enabling job history requires additional memory for the server. When the server is keeping job history, it needs 8kb-12kb of memory per job, instead of the 5kb it needs without job history. Make sure you have enough memory: multiply the number of jobs being tracked by this much memory. For example, if you are starting 100 jobs per day, and tracking history for two weeks, you’re tracking 1400 jobs at a time. On average, this will require 14.3M of memory.

- If the server is shut down abruptly, there is no loss of job information. However, the server will require longer to start up when keeping job history, because it must read in more information.
11.16 Environment Variables

The settings in $PBS_HOME/pbs_environment are available to user job scripts. You must HUP the MoM if you change the file. This file is useful for setting environment variables for `mpirun` etc.

11.17 Adjusting Job Running Time

11.17.1 Shrink-to-fit Jobs

PBS allows you or the job submitter to adjust the running time of a job to fit into an available scheduling slot. The job’s minimum and maximum running time are specified in the `min_walltime` and `max_walltime` resources. PBS chooses the actual `walltime`. Any job that requests `min_walltime` is a shrink-to-fit job.

For a complete description of using shrink-to-fit jobs, see section 4.8.41, “Using Shrink-to-fit Jobs”, on page 263.

11.18 Managing Number of Run Attempts

PBS has a built-in limit of 21 for the number of times the server can try to run a job. When the job goes over this limit, the job is held. The number of tries is recorded in the job’s `run_count` attribute. The `run_count` attribute starts at zero, and the job is held when `run_count` goes above 20.

Job submitters can set a non-negative value for `run_count` on job submission, and can use `qalter` to raise the value of `run_count`. A PBS manager or Operator can use `qalter` to raise or lower the value of `run_count`.
11.19 Allowing Interactive Jobs on Windows

1. Make sure that file and printer sharing is enabled. This is off by default.

2. Make sure that IPC$ share is enabled. You should be able to run the following command from the submission host:

   net use \<execution_host>\IPC$

   The output should look like this:

   > net use \myhost\IPC$
   c:\Users\pbsuser>net use \myhost\IPC$
   Local name
   Remote name \myhost\IPC$
   Resource type IPC
   Status Disconnected
   # Opens 0
   # Connections 1
   The command completed successfully.
Chapter 12

Administration

This chapter covers information on the maintenance and administration of PBS, and is intended for the PBS administrator. Topics covered include starting and stopping PBS, event logging, and accounting.

12.1 The PBS Configuration File

During the installation of PBS Professional, the installation script creates a configuration file named pbs.conf. This configuration file controls which daemons are to run on the local system, the directory tree location, and various runtime configuration options. Each host in a complex should have its own pbs.conf file.

12.1.1 Location of Configuration File

The configuration file is located in one of the following:

UNIX/Linux:

/etc/pbs.conf

Windows:

[PBS Destination Folder]\pbs.conf

where [PBS Destination Folder] is the path specified when PBS is installed on the Windows platform, for example:

C:\Program Files\PBS Pro\pbs.conf
or

C:\Program Files (x86)\PBS Pro\pbs.conf

12.1.2 Example of Configuration File

The following is an example of a pbs.conf file for a host which is to run the server, the scheduler, and a MoM. The server runs on the host named Host1.ExampleDomain.

PBS_EXEC=/opt/pbs/M.N.P.S
PBS_HOME=/var/spool/PBS
PBS_START_SERVER=1
PBS_START_MOM=1
PBS_START_SCHED=1
PBS_SERVER=Host1.ExampleDomain

12.1.3 Contents of Configuration File

The following table describes the available parameters for the pbs.conf configuration file:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS_BATCH_SERVICE_PORT</td>
<td>Port Server listens on</td>
</tr>
<tr>
<td>PBS_BATCH_SERVICE_PORT_DIS</td>
<td>DIS port server listens on</td>
</tr>
<tr>
<td>PBS_CORE_LIMIT</td>
<td>Limit on corefile size for PBS daemons. Can be set to an integer number of bytes or to the string &quot;unlimited&quot;.</td>
</tr>
<tr>
<td>PBS_DATA_SERVICE_PORT</td>
<td>Used to specify non-default port for connecting to data service.</td>
</tr>
<tr>
<td>PBS_ENVIRONMENT</td>
<td>Location of pbs_environment file</td>
</tr>
<tr>
<td>PBS_EXEC</td>
<td>Location of PBS bin and sbin directories</td>
</tr>
<tr>
<td>PBS_HOME</td>
<td>Location of PBS working directories</td>
</tr>
<tr>
<td>PBS_LICENSE_FILE_LOCATION</td>
<td>Location of license server.</td>
</tr>
<tr>
<td>PBS_LOCALLOG</td>
<td>Enables logging to local PBS log files</td>
</tr>
</tbody>
</table>
### Table 12-1: Parameters in `pbs.conf`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PBS_MANAGER_GLOBUS_SERVICE_PORT</code></td>
<td>Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus. Port Globus MoM listens on. Globus MoM is no longer available.</td>
</tr>
<tr>
<td><code>PBS_MANAGER_SERVICE_PORT</code></td>
<td>Port MoM listens on</td>
</tr>
<tr>
<td><code>PBS_MOM_GLOBUS_SERVICE_PORT</code></td>
<td>Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus. Port Globus MoM listens on. Globus MoM is no longer available.</td>
</tr>
<tr>
<td><code>PBS_MOM_HOME</code></td>
<td>Location of MoM working directories</td>
</tr>
<tr>
<td><code>PBS_MOM_SERVICE_PORT</code></td>
<td>Port MoM listens on</td>
</tr>
<tr>
<td><code>PBS_PRIMARY</code></td>
<td>Hostname of primary Server</td>
</tr>
<tr>
<td><code>PBS_RCP</code></td>
<td>Location of <code>rcp</code> command if <code>rcp</code> is used</td>
</tr>
<tr>
<td><code>PBS_SCHEDULER_SERVICE_PORT</code></td>
<td>Port Scheduler listens on</td>
</tr>
<tr>
<td><code>PBS_SCP</code></td>
<td>Location of <code>scp</code> command if <code>scp</code> is used; setting this parameter causes PBS to first try <code>scp</code> rather than <code>rcp</code> for file transport.</td>
</tr>
<tr>
<td><code>PBS_SECONDARY</code></td>
<td>Hostname of secondary Server</td>
</tr>
<tr>
<td><code>PBS_SERVER</code></td>
<td>Hostname of host running the Server</td>
</tr>
<tr>
<td><code>PBS_START_MOM</code></td>
<td>Set to <code>1</code> if a MoM is to run on this vnode</td>
</tr>
<tr>
<td><code>PBS_START_SCHED</code></td>
<td>Set to <code>1</code> if Scheduler is to run on this vnode</td>
</tr>
<tr>
<td><code>PBS_START_SERVER</code></td>
<td>Set to <code>1</code> if Server is to run on this vnode</td>
</tr>
<tr>
<td><code>PBS_SYSLOG</code></td>
<td>Controls use of syslog facility</td>
</tr>
<tr>
<td><code>PBS_SYSLOGSEVR</code></td>
<td>Filters syslog messages by severity</td>
</tr>
</tbody>
</table>
For information on how to use the pbs.conf file when configuring PBS for failover, see section 9.2.5.2, “Configuring the pbs.conf File”, on page 699.

12.1.4 Configuration File Caveats and Recommendations

- If you specify a location for PBS_HOME in the shell environment, make sure that this agrees with that specified in pbs.conf.
- Do not change a hostname without updating the corresponding Version 2 configuration file.
- Use a name for the server in the PBS_SERVER variable in the pbs.conf file that is not longer than 64 characters. If the short name for the server resolves to the correct host, you can use this in pbs.conf as the value of PBS_SERVER. However, if the fully-qualified domain name is required in order to resolve to the correct host, then this must be the value of the PBS_SERVER variable.

12.2 Environment Variables

PBS sets environment variables for different purposes: some variables are used by the daemons, commands, and jobs, and some environment variables are set individually for each job.

12.2.1 Environment Variables For Daemons, Commands, and Jobs

The PBS installer creates an environment file called pbs_environment. This file is used by the daemons, commands, and jobs:

- Each PBS daemon initializes its environment using this environment file
- Several commands use environment variables to determine things like the name of the default server. The environment file is useful for setting environment variables for mpirun, etc.
- Jobs inherit the contents of this environment file before they acquire settings from

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS_TMPDIR</td>
<td>Root directory for temporary files for PBS components</td>
</tr>
</tbody>
</table>

Table 12-1: Parameters in pbs.conf
.profile and .login files. Job scripts can use the environment variables set in the job’s environment.

You can edit the environment file.

12.2.1 Contents of Environment File

When this file is created, it contains the following:

- TZ=<local timezone, e.g. US/Pacific>
- PATH=/bin:/usr/bin

For a list of PBS environment variables, see “PBS Environment Variables” on page 459 of the PBS Professional Reference Guide.

To support X forwarding, edit MoM’s PATH variable to include the directory containing the xauth utility.

12.2.1.2 Location of Environment File

The PBS environment file is located here:

PBS_HOME/pbs_environment

12.2.1.3 Environment File Requirements

You must restart each daemon after making any changes to the environment file.

12.2.1.4 Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

12.2.2 Job-specific Environment Variables

For each job, the qsub command creates environment variables beginning with PBS_O_, and puts them in the job’s environment. They are not written to pbs_environment. The server sets some of these environment variables if the qsub command does not set them.

For each job, the MoM on the primary execution host creates a file of the hosts to be used by the job. The node file is put in the job’s environment, but the host list is not written to pbs_environment. The location of the node file is specified in the PBS_NODEFILE environment variable, which is set for the job only. See "The Job’s Node File", on page 106 of the PBS Professional User’s Guide.
Some environment variables are set by commands. The PBS `mpiexec` script sets `PBS_CPUSET_DEDICATED`.

For a list of environment variables used and set by the `qsub` command, see "Environment Variables" on page 229 of the PBS Professional Reference Guide.

## 12.3 The Accounting Log

The PBS Server maintains an accounting log. This file is maintained on the Server host only; it is not written on the execution hosts. The log name defaults to `PBS_HOME/server_priv/accounting/ccyymmdd` where `ccyymmdd` is the date. The accounting log files may be placed elsewhere by specifying the `-A` option on the `pbs_server` command line. The option argument is the full (absolute) path name of the file to be used. If a null string is given, then the accounting log will not be opened and no accounting records will be recorded. For example

```
pbs_server -A ""
```

The accounting file is changed according to the same rules as the event log files. If the default file is used, named for the date, the file will be closed and a new one opened every day on the first event (write to the file) after midnight. With either the default file or a file named with the `-A` option, the Server will close the accounting log upon daemon/service shutdown and reopen it upon daemon/service startup.

On UNIX the Server will also close and reopen the account log file upon the receipt of a `SIGHUP` signal. This allows you to rename the old log and start recording again on an empty file. For example, if the current date is February 9, 2005 the Server will be writing in the file `20050209`. The following actions will cause the current accounting file to be renamed `feb9` and the Server to close the file and start writing a new `20050209`:

```
cd $PBS_HOME/server_priv/accounting
mv 20050209 feb9
kill -HUP 1234     (the Server’s pid)
```

On Windows, to manually rotate the account log file, shut down the Server, move or rename the accounting file, and restart the Server. For example, to cause the current accounting file to be renamed `feb9` and the Server to close the file and start writing a new `20050209`:

```
\cd "%PBS_HOME\server_priv\accounting"
\net stop pbs_server
\move 20050209 feb9
\net start pbs_server
```
12.3.1 How To Find Accounting Information

Accounting logs are written on the Server host only.

12.3.1.1 All Log Files

To get information about a job that is running or has finished, use the `tracejob` command at the Server host and any execution hosts on which the job ran. The `tracejob` command looks at all log files.

```
tracejob <job ID>
```

See “`tracejob` on page 234 of the PBS Professional Reference Guide”, or the `tracejob(8B)` man page for details about using the `tracejob` command.

12.3.1.2 Server and Accounting Log Files

To get information about a job that has finished, use the `pbs-report` command at the Server host. The `pbs-report` command looks at Server logs and accounting logs.

```
pbs-report [options]
```

See “`pbs-report` on page 33 of the PBS Professional Reference Guide” or the `pbs-report(8B)` man page for details about using the `pbs-report` command.

12.3.2 Accounting Log Contents

The PBS accounting file is a text file with each entry terminated by a newline. There is no limit to the size of an entry.

12.3.2.1 Log Entry Format

See “`Log Entry Format` on page 419 of the PBS Professional Reference Guide” for a description of the accounting log’s format.

12.3.2.2 Record Types

See “`Record Types` on page 420 of the PBS Professional Reference Guide” for a description of accounting log record types.
12.3.2.3 Logging Resource Usage

MoM periodically polls jobs for usage by the jobs running on her host, collects the results, and reports this to the server. When a job exits, she polls again to get the final tally of usage for that job.

For example, MoM polls the running jobs at times T1, T2, T4, T8, T16, T24, and so on.

The output shown by a `qstat` during the window of time between T8 and T16 shows the resource usage up to T8.

If the `qstat` is done at T17, the output shows usage up through T16. If the job ends at T20, the accounting log (and the final log message, and the email to the user if "qsub -me" was used in job submission) contains usage through T20.

The final report does not include the epilogue. The time required for the epilogue is treated as system overhead.

12.3.3 PBS Accounting and Windows

PBS will save information such as username, group name, and account name in the accounting logs found in `PBS_HOME\server_priv\accounting`. Under Windows, these saved entities can contain space characters, thus PBS will put a quote around string values containing spaces. For example,

```
user=pbstest group=None account="Power Users"
```

Otherwise, one can specify the replacement for the space character by adding the `-s` option to the `pbs_server` command line option. This can be set as follows:

2. Select PBS_SERVER.
3. Stop the Server
4. Specify in start parameters the option for example “-s %20”.
5. Start the Server

This will replace space characters as “%20” in `user=, group=, account= entries in accounting log file`:

```
user=pbstest group=None account=Power%20Users
```
Note: If the first character of the replacement string argument to -s option appears in the input string itself, then that character will be replaced by its hex representation prefixed by %. For example, given:

    account=Po%wer Users

Since % also appears the above entry and our replacement string is “%20”, then replace this % with its hex representation (%25):

    account="Po%25wer%20Users"

12.4 Event Logging

PBS provides event logging for the Server, the Scheduler, and each MoM. You can use log-files to monitor activity in the PBS complex.

12.4.1 PBS Events

The amount and type of output in the PBS event logfiles depends on the specified log filters for each component. All three PBS components can be directed to record only messages pertaining to certain event types, called event classes. The specified event classes are logically “or-ed” to produce a mask representing the events to be logged by the Server or MoM, or the events to be excluded by the scheduler. The hexadecimal value for each event class is shown in section 12.4.4, “Log Event Classes”, on page 868. When events appear in the log file, they are tagged with their hexadecimal value, without a preceding “0x”.

12.4.2 Event Logfiles

Each PBS daemon writes a separate event logfile. By default, each daemon writes a file that has the current date as its name in the PBS_HOME/<component>_logs directory. The location of the logfile can be overridden with the -L option to each daemon’s command. For example, to override the Server’s logfile location:

    pbs_server  -L <new path>

Each daemon closes the day’s log file and opens a new log file on the first message written after midnight. If no messages are written, the old log file stays open. Each daemon closes and reopens the same logfile when the daemon receives a SIGHUP.
Each daemon writes its version and build information to its event logfile each time it is started or restarted, and also when the logfile is automatically rotated out. The version and build information appear in individual records. These records contain the following substrings:

```
    pbs_version = <PBSPro_stringX.stringY.stringZ.5-digit seq>

    build = <status line from config.status, etc>
```

Example:

```
    pbs_version = PBSPro_9.2.0.63106
    build = '--set-cflags=-g -O0' --enable-security=RCRYPT ...
```

If the daemon cannot write to its log file, it writes the error message to the console. Some errors that appear before the daemon has backgrounded itself may appear on standard error.

The maximum number of characters in the message portion of a log entry is 4096.

### 12.4.3 Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

### 12.4.4 Log Event Classes

PBS allows specification of the types of events that are logged for each daemon. Each type of log event has a different log event class. All daemons use the same log event class for the same type of event. The following table lists the log event class for each type of event.

<table>
<thead>
<tr>
<th>Name</th>
<th>Decimal</th>
<th>Hex</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBSEVENT_ERROR</td>
<td>1</td>
<td>0x0001</td>
<td>Internal PBS errors</td>
</tr>
<tr>
<td>PBSEVENT_SYSTEM</td>
<td>2</td>
<td>0x0002</td>
<td>System (OS) errors, such as malloc failure</td>
</tr>
<tr>
<td>PBSEVENT_ADMIN</td>
<td>4</td>
<td>0x0004</td>
<td>Administrator-controlled events, such as changing queue attributes</td>
</tr>
<tr>
<td>PBSEVENT_JOB</td>
<td>8</td>
<td>0x0008</td>
<td>Job related events, e.g. submitted, ran, deleted</td>
</tr>
</tbody>
</table>
12.4.4.1 Specifying Log Event Classes

Each daemon uses an integer representation of a bit string to specify its log event classes. The bit string can be decimal (or hexadecimal, for the MoM and the scheduler). Each daemon’s log event classes are specified in a different way: the Server and the MoMs use a bit string that includes the events to be logged, whereas the Scheduler’s bit string represents the events that are to be excluded from its log.

For example, if you want the Server to log all events except those at event classes 512 and 1024 (hex 0x200 and 0x400), you would use a log event class of 511. This is $256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1$. If you want to log events at event classes 1, 2, and 16, you would set the log event class to 19.

### Table 12-2: PBS Events and Log Event Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Decimal</th>
<th>Hex</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBSEVENT_JOB_USAGE</td>
<td>16</td>
<td>0x0010</td>
<td>Job resource usage</td>
</tr>
<tr>
<td>PBSEVENT_SECURITY</td>
<td>32</td>
<td>0x0020</td>
<td>Security related events</td>
</tr>
<tr>
<td>PBSEVENT_SCHED</td>
<td>64</td>
<td>0x0040</td>
<td>When the Scheduler was called and why</td>
</tr>
<tr>
<td>PBSEVENT_DEBUG</td>
<td>128</td>
<td>0x0080</td>
<td>Common debug messages</td>
</tr>
<tr>
<td>PBSEVENT_DEBUG2</td>
<td>256</td>
<td>0x0100</td>
<td>Debug event class 2</td>
</tr>
<tr>
<td>PBSEVENT_RESV</td>
<td>512</td>
<td>0x0200</td>
<td>Reservation-related messages</td>
</tr>
<tr>
<td>PBSEVENT_DEBUG3</td>
<td>1024</td>
<td>0x0400</td>
<td>Debug event class 3. Debug messages rarer than event class 2.</td>
</tr>
<tr>
<td>PBSEVENT_DEBUG4</td>
<td>2048</td>
<td>0x0800</td>
<td>Debug event class 4. Limited messages.</td>
</tr>
</tbody>
</table>
The following table shows the log event parameter for each daemon:

<table>
<thead>
<tr>
<th>PBS</th>
<th>Parameter/Attribute</th>
<th>Reference</th>
<th>How to Make Parameter Take Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>log_events attribute</td>
<td>“Server Attributes” on page 316 of the PBS Professional Reference Guide</td>
<td>Takes effect immediately with qmgr</td>
</tr>
<tr>
<td>MoM</td>
<td>$logevent parameter</td>
<td>“Contents of MoM Configuration File” on page 268 of the PBS Professional Reference Guide</td>
<td>Requires SIGHUP to MoM</td>
</tr>
<tr>
<td>Scheduler</td>
<td>log_filter parameter</td>
<td>“Configuration Parameters” on page 282 of the PBS Professional Reference Guide</td>
<td>Requires SIGHUP to Scheduler</td>
</tr>
</tbody>
</table>

When reading the PBS event logfiles, you may see messages of the form “Type 19 request received from PBS_Server...”. These “type codes” correspond to different PBS batch requests.

12.4.4.1.i Specifying Server Log Events

The Server’s log events are specified in the Server’s log_events attribute. The attribute is an integer representation of a bit string, where the integer includes all events to be logged. To set the value, use the qmgr command:

```
Qmgr: set server log_events = <value>
```

For example, to log only debug event class 3 (1024, or 0x0400) and internal PBS errors (1, or 0x0001), set the value to 1025 (1024 +1, or 0x0401). To included all events, set the value to 2047 (or -7). The default value for this attribute is 511 (0x1ff). It can be set by Operators and Managers only. See “Server Attributes” on page 316 of the PBS Professional Reference Guide.

12.4.4.1.ii Specifying MoM Log Events

Each MoM’s log events are specified in the $logevent parameter in that MoM’s configuration file PBS_HOME/mom_priv/config. The parameter is an integer representation of a bit string, where the integer includes all events to be logged. For example, to log only debug
event class 3 (1024, or 0x0400) and internal PBS errors (1, or 0x0001), set the value to 1025 (1024 +1, or 0x0401). To set the value, add the $logevent line in PBS_HOME/mom_priv/config. To include all events, set the value to 4095 (0xffffffff). The default value used by MoM is 975 (0x03cf). This parameter can be set by root only. See “Contents of MoM Configuration File” on page 268 of the PBS Professional Reference Guide.

12.4.4.1.iii Specifying Scheduler Log Events

The Scheduler’s log events are specified in the log_filter Scheduler configuration parameter in the Scheduler configuration file PBS_HOME/sched_priv/sched_config. This parameter is a bit string that is made up of all the log events to be excluded from the Scheduler’s log file. To set the value, edit the log_filter line in PBS_HOME/sched_priv/sched_config. For example, to exclude common debug messages (128, or 0x0080) and reservation-related messages (512, or 0x0200), set the parameter value to 640 or 0x0280 (128 + 512, or 0x0200 + 0x0080). To include all events, set the value to 0 (zero). To log nothing, set the value to 4095 (0xffffffff). (The default value is 3328 (0x0D00). This parameter can be set by root only. See “Configuration Parameters” on page 282 of the PBS Professional Reference Guide.

12.4.5 Event Logfile Format and Contents

12.4.5.1 Event Logfile Format

Each component event logfile is a text file with each entry terminated by a new line. The format of an entry is:

logfile-date-time;event_code;server_name;object_type;object_name;message

- The logfile-date-time field is a date and time stamp in the format:

  mm/dd/yyyy hh:mm:ss

- The event_code is a bitmask for the type of event which triggered the event logging. It corresponds to the bit position, 0 to n, of each log event in the event mask of the PBS component writing the event record. See section 12.4.1, “PBS Events”, on page 867 for a description of the event mask.

- The server_name is the name of the Server which logged the message. This is recorded in case a site wishes to merge and sort the various logs in a single file.

- The object_type is the type of object which the message is about. All messages are asso-
associated with an object_type. The following lists each possible object_type:

**Table 12-4: List of Event Logfile Object Types**

<table>
<thead>
<tr>
<th>object_type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Svr</td>
<td>for server</td>
</tr>
<tr>
<td>Que</td>
<td>for queue</td>
</tr>
<tr>
<td>Job</td>
<td>for job</td>
</tr>
<tr>
<td>Req</td>
<td>for request</td>
</tr>
<tr>
<td>Fil</td>
<td>for file</td>
</tr>
<tr>
<td>Act</td>
<td>for accounting string</td>
</tr>
<tr>
<td>Node</td>
<td>for vnode or host</td>
</tr>
<tr>
<td>Resv</td>
<td>for reservation</td>
</tr>
<tr>
<td>Sched</td>
<td>for scheduler</td>
</tr>
</tbody>
</table>

- The object_name is the name of the specific object.
- The message field is the text of the log message.

### 12.4.5.2 Scheduler Commands

These commands tell the scheduler why a scheduling cycle is being started. These commands appear in the Server’s logfile. Each has a decimal value, shown below. The following table shows commands from the Server to the scheduler.

**Table 12-5: Commands from Server to Scheduler**

<table>
<thead>
<tr>
<th>Value</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>New job enqueued</td>
</tr>
<tr>
<td>2</td>
<td>Job terminated</td>
</tr>
<tr>
<td>3</td>
<td>Scheduler time interval reached</td>
</tr>
<tr>
<td>4</td>
<td>Cycle again after scheduling one job</td>
</tr>
</tbody>
</table>
### Table 12-5: Commands from Server to Scheduler

<table>
<thead>
<tr>
<th>Value</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Scheduling command from operator or manager</td>
</tr>
<tr>
<td>7</td>
<td>Configure</td>
</tr>
<tr>
<td>8</td>
<td>Quit (qterm -s)</td>
</tr>
<tr>
<td>9</td>
<td>Ruleset changed</td>
</tr>
<tr>
<td>10</td>
<td>Schedule first</td>
</tr>
<tr>
<td>11</td>
<td>A reservation’s start time has been reached</td>
</tr>
<tr>
<td>12</td>
<td>Schedule a job (qrun command has been given)</td>
</tr>
<tr>
<td>13</td>
<td>Stopped queue is started</td>
</tr>
<tr>
<td>14</td>
<td>Job moved into local queue (queue at this server)</td>
</tr>
<tr>
<td>15</td>
<td>eligible_time_enable is turned on</td>
</tr>
<tr>
<td>16</td>
<td>PBS attempting to reconfirm degraded reservation</td>
</tr>
</tbody>
</table>

### 12.4.6 Logging Job Usage

PBS can log per-vnode cputime usage. The mother superior logs cputime in the format “hh:mm:ss” for each vnode of a multi-vnode job. The log event class of these messages is 0x0100.

Under UNIX/Linux, to append job usage to standard output for an interactive job, use a shell script for the epilogue which contains the following:

```bash
#!/bin/sh
tracejob -sl $1 | grep 'cput'
```

This behavior is not available under Windows.
12.4.7 Managing Log Files

12.4.7.1 Disk Space for Log Files

It is important not to run out of disk space for logging. You should periodically check the available disk space, and check the size of the log files PBS is writing, so that you know how fast you are using up disk space. Make sure that you always have more than enough disk space available for log files.

12.4.7.2 Dividing Up Log Files

You may wish to divide a day’s logging up into more than one file. You may want to create a logfile that contains only the entries of interest. You can specify a file for a daemon’s event log. See section 12.4.7.3, “Specifying Log File Path”, on page 875. The next sections describe how to break up your log files.

12.4.7.2.i Dividing Log Files on UNIX/Linux

On UNIX and Linux systems, all daemons close and reopen the same named log file when they are sent a SIGHUP. The process identifier (PID) of each daemon is available in its lock file in its home directory. You can move the current log file to a new name and send SIGHUP to restart the file using the following commands:

```bash
cd $PBS_HOME/<daemon>_logs
mv <current log file> <archived log file>
kill -HUP 'cat ../<daemon>_priv/<daemon>.lock'
```

12.4.7.2.ii Dividing Log Files on Windows

On Windows systems, you can rotate the event log files by stopping the service for which you want to rotate the logfile, moving the file, and then restarting that service. For example:

```bash
cd "%PBS_HOME%\<service>_logs"
net stop pbs_<service>
mv <current log file> <archived log file>
net start pbs_<service>
```
### 12.4.7.3 Specifying Log File Path

You may wish to specify an event logfile path that is different from the default path. Each daemon has an option to specify a different path for the daemon’s event logfile. This option is the `-L logfile` option, and it is the same for all daemons. For example, to start the Scheduler so that it logs events in `/scratch/my_sched_log`:

```bash
pbs_sched -L /scratch/my_sched_log
```

See the `pbs_server(8B)`, `pbs_sched(8B)`, and `pbs_mom(8B)` manual pages.

### 12.4.8 Extracting Logged Information

You can use the `tracejob` command to extract information from log files, such as why a job is not running or when a job was queued. The `tracejob` command can read both event logs and accounting logs. See the `tracejob(8B)` manual page.

### 12.5 Using the UNIX syslog Facility

Each PBS component logs various event classes of information about events in its own log file. While having the advantage of a concise location for the information from each component, the disadvantage is that in a complex, the logged information is scattered across each execution host. The UNIX syslog facility can be useful.

If your site uses the `syslog` subsystem, PBS may be configured to make full use of it. The following entries in `pbs.conf` control the use of `syslog` by the PBS components:

<table>
<thead>
<tr>
<th>PBS_LOCALLOG</th>
<th>Enables logging to local PBS log files. Only possible when logging via <code>syslog</code> feature is enabled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td><code>PBS_LOCALLOG=x</code> enables local logging. Only possible when the <code>syslog</code> feature is enabled.</td>
</tr>
<tr>
<td>0</td>
<td><code>PBS_LOCALLOG=0</code> enables no local logging.</td>
</tr>
<tr>
<td>1</td>
<td><code>PBS_LOCALLOG=1</code> enables local logging.</td>
</tr>
</tbody>
</table>

See the `pbs_server(8B)`, `pbs_sched(8B)`, and `pbs_mom(8B)` manual pages.
### Table 12-6: Entries in *pbs.conf* for Using Syslog

<table>
<thead>
<tr>
<th>PBS_SYSLOG=x</th>
<th>Controls the use of syslog and syslog “facility” under which the entries are logged. If x is:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - no syslogging</td>
<td></td>
</tr>
<tr>
<td>1 - logged via LOG_DAEMON facility</td>
<td></td>
</tr>
<tr>
<td>2 - logged via LOG_LOCAL0 facility</td>
<td></td>
</tr>
<tr>
<td>3 - logged via LOG_LOCAL1 facility</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>9 - logged via LOG_LOCAL7 facility</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PBS_SYSLOGSEVR=y</th>
<th>Controls the severity level of messages that are logged; see <code>/usr/include/sys/syslog.h</code>. If y is:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - only LOG_EMERG messages are logged</td>
<td></td>
</tr>
<tr>
<td>1 - messages up to LOG_ALERT are logged</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>7 - messages up to LOG_DEBUG are logged</td>
<td></td>
</tr>
</tbody>
</table>

**IMPORTANT:**

PBS_SYSLOGSEVR is used in addition to PBS's log_events mask which controls the class of events (job, vnode, ...) that are logged.

## 12.6 Managing Machines

### 12.6.1 Offlining Hosts and Vnodes

To offline an entire host, use the `pbsnodes` command. Use the name of the natural vnode, which is usually the name of the host:

```bash
pbsnodes -o <name of natural vnode>
```

All vnodes on this host are offline.

To offline a single vnode, use the `qmgr` command, with the name of the vnode:

```bash
qmgr -c "set node foo[3] state=offline"
```
12.6.1.1 Caveats of Offlining

If you set a vnode with no running jobs **offline**, the server will not attempt to communicate with the vnode. Therefore, the server will not notice that the vnode is up until you clear the **offline** state. For example, a vnode that is both **down** and **offline** will not be marked up by the server until you clear the **offline** state.

12.6.2 Changing Hostnames or IP Addresses

Do not change hostname or IP address when jobs are running.

To change a hostname or IP address:
1. Make sure no jobs are running
2. Stop all PBS daemons
3. Make a backup of **PBS_HOME**
4. Change the hostname or IP address
5. Restart all PBS daemons
6. If a host has a corresponding Version 2 configuration file, make sure that it is consistent with the new hostname
7. If you are running **nscd**, restart **nscd** on all hosts

12.6.3 Discovering Server’s Last Reboot Time

Under UNIX/Linux, you can find the timestamp of the most recent time PBS started up in **/var/tmp/pbs_boot_check**.

The permission of this file is set to 0644; only the PBS init script should modify this file. Do not modify this file. If you do so, you violate the configuration requirements of PBS.

This file is not available under Windows.

12.7 Managing the Data Service

12.7.1 PBS Monitors Data Service

PBS monitors its connection to the data service. If the connection is broken (for example, because the data service is down), PBS tries to reestablish the connection. If necessary, PBS restarts the data service.
If failover is configured, and PBS cannot reestablish a connection, PBS quits.
If failover is not configured, PBS attempts to reestablish the connection until it succeeds.
When the server is stopped, it stops the data service.

12.7.2 Data Service on UNIX/Linux

The data service daemon requires, and runs from, the data service user account. On UNIX, the requirements for this account are the following:

- Non-root account
- Account is enabled
- If you are using failover, the UID of this account must be the same on both primary and secondary server hosts
- We recommend that the account is called pbsdata. The installer looks for an account called pbsdata. If this account exists, the installer does not need to prompt for a username, and can install silently.
- Root must be able to su to the data service account and run commands as that user. Do not add lines such as ‘exec bash’ to the .profile of the data service account. If you want to use bash or similar, set this in the /etc/passwd file, via the OS tools for user management.
- The data service account must have a home directory.

12.7.3 Data Service on Windows

On Windows, this account is the PBS service account from which the other PBS daemons run. The data service account must be the same as the PBS service account.
The data service account must have a home directory.

12.7.4 Data Service Account and Password

The default password generated at install time is a random password which is known only to the PBS server.
Changing the password is necessary only if you want to manually log into the database to check data or change something. Otherwise it is not necessary.
12.7.4.1 Setting Data Service Account and Password

You can change the user account and/or password for the PBS data service using the `pbs_ds_password` command. Use this command if you need to change the user account or update the password for the data service. You must be root or administrator to run the `pbs_ds_password` command. See “`pbs_ds_password`” on page 47 of the PBS Professional Reference Guide.

To change the data service user account:

```
pbs_ds_password -C <new user account>
```

To change the data service password:

```
pbs_ds_password
```

12.7.4.2 Caveats

- The specified new user account must already exist.
- The account name cannot be changed while the data service is running.
- Do not delete `PBS_HOME/server_priv/db_password`. Doing so will prevent the `pbs_ds_password` command from being able to function.
- Do not change the data service password using any method other than the `pbs_ds_password` command.

12.7.5 Starting and Stopping the Data Service

PBS automatically starts and stops the data service. However, you can start, stop, or check the status of the PBS data service using the `pbs_dataservice` command. See “`pbs_dataservice`” on page 46 of the PBS Professional Reference Guide.

To start the data service:

```
pbs_dataservice start
```

To stop the data service:

```
pbs_dataservice stop
```

To get the status of the data service:

```
pbs_dataservice status
```

12.7.5.1 Caveats for Starting and Stopping Data Service

- Do not start or stop the data service using anything except the `pbs_dataservice`
command. Start or stop the data service using only the `pbs_dataservice` command.

- The data service cannot be stopped while the PBS server is running.

12.7.6 Changing Data Service Port

You can change the port that the data service listens on by changing the setting of the PBS_DATA_SERVICE_PORT entry in pbs.conf.

12.7.6.1 Caveats

- The PBS daemons must not be running when the port is changed.
- The data service must not be running when the port is changed.

12.7.7 File Ownership

The files under `PBS_HOME/datastore` are owned by the data service user account.

12.8 Enabling Passwordless Authentication

You must enable passwordless authentication so that job files can be staged in and out. You must also choose and set a file transfer mechanism such as `rcp` or `scp` for remote file copying. Before you set up the remote file copy mechanism, enable passwordless authentication for it.

Enable passwordless authentication for each machine in the complex, and for any machine from which or to which files will be transferred.

You can use any authentication method you want, such as a `hosts.equiv` file, an authorized keys file, or `.rhosts` authentication. You can choose a cipher and use encryption; balance the CPU time required by encryption with the CPU time required by MoMs and job tasks.

PBS requires that `rsh/rcp` and/or `ssh/scp` works between each pair of hosts where files will be transferred. Test whether you have succeeded by logging in as root, and using your chosen file transfer mechanism to copy a file between machines.
12.9 Setting File Transfer Mechanism

12.9.1 File Transfer in PBS

MoM does the work of transferring files, using the mechanism you specify. MoM transfers files when she stages them in or out for a job, and when she delivers output and error files. MoM always tries to determine whether the source and destination for a file transfer are both local. If they are, she uses the local copy mechanism (/bin/cp on UNIX/Linux, and xcopy on Windows). You can use the $usecp MoM configuration parameter to tell MoM which local directories are mapped to mounted directories, so that she can use the local copy mechanism.

For remote copying on UNIX/Linux, PBS uses rcp by default; you can configure PBS to use scp for this. You can also tell PBS to use any script or command for remote file transfer, such as rsync, gsiftp, etc.

For remote copying on Windows, PBS ships with and uses by default a version of rcp called pbs_rcp.

PBS does not impose limitations on the size of files being transferred. Any limitations are caused by the commands themselves. The pbs_rcp command should be as fast as other implementations of rcp.

12.9.1.1 Configuration Parameters Affecting File Transfer

You configure MoM’s file transfer mechanisms using the following:

- Local copy: the $usecp MoM configuration parameter
- Remote copy: the PBS_RCP entry in pbs.conf, and the PBS_SCP entry in pbs.conf

12.9.1.2 How MoM Chooses File Transfer Method

If MoM knows that she is performing a local file transfer, she uses her local copy mechanism.

If MoM is transferring a remote file, she chooses according to the following:

- If no pbs.conf parameters are defined, MoM uses rcp.
- If a command is specified in PBS_RCP, and PBS_SCP is not defined, MoM uses the command specified in PBS_RCP.
- If a command is specified in PBS_SCP, MoM uses the command specified in PBS_SCP.
12.9.1.2.i  When Multiple Attempts Are Required

If necessary, MoM tries to transfer a file multiple times, with an increasing delay between each attempt:

- If MoM is using her local copy mechanism, she tries it up to four times
- If MoM is using rcp, pbs_rcp, or the entry in PBS_RCP, she tries it up to four times
- If MoM is using the entry in PBS_SCP:
  - She first tries this, and if it fails, she tries rcp, pbs_rcp, or the entry in PBS_RCP if it is configured
  - She repeats this sequence four times

12.9.1.3  Options Passed to File Transfer Commands

12.9.1.3.i  Options Passed on UNIX/Linux

MoM automatically uses these options on UNIX/Linux:

**Table 12-1: File Transfer Mechanism Options on UNIX/Linux**

<table>
<thead>
<tr>
<th>Distance</th>
<th>Mechanism</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote</td>
<td>PBS_RCP entry or rcp</td>
<td>-rp</td>
</tr>
<tr>
<td>Remote</td>
<td>PBS_SCP entry</td>
<td>-Brvp</td>
</tr>
<tr>
<td>Local</td>
<td>/bin/cp</td>
<td>-rp</td>
</tr>
</tbody>
</table>

12.9.1.3.ii Options Passed on Windows

MoM automatically uses these options on Windows:

**Table 12-2: File Transfer Mechanism Options on Windows**

<table>
<thead>
<tr>
<th>Distance</th>
<th>Mechanism</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote</td>
<td>PBS_RCP entry, rcp, or pbs_rcp</td>
<td>-E -r</td>
</tr>
<tr>
<td>Remote</td>
<td>PBS_SCP entry</td>
<td>-Brv</td>
</tr>
<tr>
<td>Local</td>
<td>xcopy</td>
<td>/e/i/q/y</td>
</tr>
</tbody>
</table>
12.9.2  Configuring MoM for Local Copy

MoM uses her local copy mechanism whenever she knows she will perform a local copy. To tell her which directories can be treated as local, specify the mappings between local and mounted directories in MoM’s $usecp configuration parameter.

12.9.2.1  Configuring the $usecp MoM Parameter

This tells MoM where to look for files in a shared file system, so that she can use the local copy agent for these files. This is useful when you have common mount points across execution hosts.

Format:

$usecp <hostname> :<source_prefix> <destination_prefix>

Both source_prefix and destination_prefix are absolute pathnames of directories, not files.

You can use a wildcard ("*") as the first element only, to replace hostname.

MoM uses a local copy mechanism to transfer files when staging or delivering output, under the following circumstances:

• The destination is a network mounted file system
• The source and destination are both on the local host
• The source_prefix can be replaced with the destination_prefix on hostname

You can map multiple directories. Use one line per mapping.

You must HUP MoM after making this change.

12.9.2.1.i  UNIX/Linux and $usecp

MoM uses /bin/cp for the local copy mechanism on UNIX/Linux.

Format:

$usecp <hostname> :<source_directory> <destination_directory>

Use trailing slashes on both the source and destination directories.

Example 12-1: Configuring $usecp on UNIX/Linux:

$usecp */home/*home/
$usecp *.example.com:/home/ /home/
$usecp */home/user */home/user
$usecp */data */data
$usecp HostA:/users/work/myproj/ /sharedwork/proj_results/
12.9.2.1.ii Windows and $usecp
MoM uses xcopy for the local copy mechanism on Windows.

Format:

$usecp <host name>:<drive name>:<directory> <drive name>:<directory>

When a network location is mapped to a local drive, you can cover all host names and case-sensitivity using entries similar to these:

$usecp *:Q: Q:
$usecp *:q: q:

Using this mapping, when MoM sees files with this format:

<hostname>:Q:file-path

or

<hostname>:q:file-path

she passes them to the copy command with this format:

Q:file-path

or

q:file-path

Example 12-2: Mapping locations with different directory names:

$usecp HostB:C:/xxxxx C:/yyyyy

12.9.3 Configuring MoM for Remote Copy

12.9.3.1 Configuring MoM to use rcp, pbs_rcp or PBS_RCP Entry

If you want MoM to use rcp on UNIX/Linux, or pbs_rcp on Windows, you do not need to do anything.

If you want MoM to use a different rcp, or another mechanism such as a script:

1. Make sure that rcp and rsh are installed on each host involved in the file transfer.

2. Specify the absolute path to the command or script in the PBS_RCP entry in pbs.conf.

3. If MoM is running, HUP MoM.
The **PBS_RCP** `pbs.conf` entry is the absolute path to a command or script used for remote transfer. If MoM is unable to copy using the **PBS_SCP** entry, she uses the entry in **PBS_RCP** as an alternate method.

MoM calls the command this way:

```
$PBS_RCP -rp <path to source> <user>@<destination.host>:<path to destination>
```

You cannot specify options inside the **PBS_RCP** entry.

### 12.9.3.2 Configuring MoM to use `scp` or **PBS_SCP** Entry

If you want MoM to use `scp` for remote copying, follow the steps below:

1. Make sure that `scp` and `ssh` are installed on each host involved in the file transfer.
2. If you use plain `scp` without a wrapper script, MoM calls it with the `-B` option, which requires passwordless authentication. If you have not done so, set up passwordless authentication on all machines involved in file transfer. See section 12.8, “Enabling Passwordless Authentication”, on page 880.
3. Set **PBS_SCP** to the absolute path to `scp`.
4. If the MoM is already running, HUP the MoM.

The **PBS_SCP** `pbs.conf` entry is the absolute path to a command or script used for remote transfer. **PBS_SCP** overrides **PBS_RCP**. When this entry is defined, this entry overrides **PBS_RCP**, and MoM tries this entry first for remote transfers.

MoM calls the command this way:

```
$PBS_SCP -Brvp <path to source> <user>@<destination.host>:<path to destination>
```

You cannot specify options inside the **PBS_SCP** entry.

### 12.9.3.3 Configuring MoM to Use Different Flags, a Script, or a Different Command

If you want MoM to use different flags to `rcp` or `scp`, or a different command, or your own script, for remote file transfer:

1. If needed, write a script that does what you need
2. Specify the path to the command or script in **PBS_SCP** in `pbs.conf`
3. If the MoM is already running, HUP the MoM.
When MoM calls PBS_SCP, she calls it with the -Brvp (UNIX/Linux) or -Brv (Windows) flags. This means that when you are writing a script, the arguments being passed to the script are:

- $1: -Brvp or -Brv
- $2: path to source
- $3: path to destination

You choose which arguments the script passes to the command inside the script. If you are using a different command, make sure that you pass the correct flags to it.

Example 12-3: Pass desired options to scp by writing a wrapper script for scp that contains the desired options, and pointing PBS_SCP to the wrapper script. In this case, we don’t use the default -Brvp, which is passed to the script as $1. The script does not pass $1 to scp; instead, it specifies -Br. We do pass in the source and destination as $2 and $3.

In pbs.conf:

```
PBS_SCP=/usr/bin/scp_pbs
```

In /usr/bin/scp_pbs:

```
#!/bin/sh
/usr/bin/scp -Br $2 $3
```

Example 12-4: Use rsync by writing a wrapper script that passes all arguments except for the first (-Brvp) to rsync, and pointing PBS_SCP to the wrapper script. In this case, the script passes all but the first argument to rsync as $*.

We get rid of the first argument using the `shift` command.

In pbs.conf:

```
PBS_SCP=/usr/bin/rsync_pbs
```

In /usr/bin/rsync_pbs:

```
#!/bin/sh
shift
shift
/usr/bin/rsync -avz -e ssh $*
```

For remote copying, MoM tries the PBS_SCP entry in pbs.conf first. If you configure both PBS_RCP and PBS_SCP with scripts or commands, put the script or command that you want MoM to try first in PBS_SCP.
12.9.4 Troubleshooting File Transfer

When using `rcp`, the copy of output or staged files can fail for the following reasons:

- The user lacks authorization to access the specified system
- Under UNIX, if the user’s `.cshrc` prints any characters to standard output, e.g. contains an `echo` command, the copy will fail

Local and remote delivery of output may fail for the following additional reasons:

- A directory in the specified destination path does not exist
- A directory in the specified destination path is not searchable by the user
- The target directory is not writable by the user

12.9.5 Advice on Improving Performance

12.9.5.1 Avoiding Server Host Overload

Avoid staging files from the server host, unless you can isolate the daemons from the effects of CPU and memory usage by `scp/ssh`, by using a mechanism such as `cpuset`. Consider the impact from a large job array that causes many files to be staged from the server host. Instead, use a shared filesystem. See section 12.9.5.2, “Avoiding Remote Transfers in Large Complexes”, on page 887.

12.9.5.2 Avoiding Remote Transfers in Large Complexes

If you are running a very large HPC complex, consider using MoM’s `$usecp` directive to avoid `rcp` and `scp` transfers. Instead, have your users place input files on a shared filesystem before submitting jobs, write their output to the shared filesystem, and keep as much as possible out of `stdout` and `stderr`.

12.9.5.3 Improving Performance for `ssh`

If network bandwidth is a limiting factor, you can use compression to improve performance. However, if CPU usage and/or memory are limiting factors, do not use compression, because compression also requires CPU and memory.

You can use compression ciphers that minimize the CPU and memory load required, for example `arcfour` or `blowfish-cbc`:

```
ciphers       arcfour,blowfish-cbc
```
12.9.5.4 Improving Performance when Staging Similar Files

If you are staging in many similar files, for example, for job arrays, you can use `rsync` in a wrapper script. Follow the instructions in section 12.9.3.3, “Configuring MoM to Use Different Flags, a Script, or a Different Command”, on page 885.

12.9.5.5 Avoiding Limits on `ssh` Connections

To prevent `scp` requests being denied when using `ssh`, you can set higher limits on incoming `ssh` connections. By default `ssh` is configured to treat more than 10 incoming connections (plus 10 in the authentication phase) as a denial-of-service attack, even on machines that could service many more requests.

Set higher limits in `/etc/ssh/sshd_config` for servers that are meant to service a lot of incoming openSSH sessions, but only on machines that have enough CPU and memory to service all of the requests.

Set these limits for servers that are meant to service a lot of incoming openSSH sessions, but only on machines that have enough CPU and memory to service all of the requests.

See the `MaxSessions` and `MaxStartups` parameters in the man page for `sshd_config`. You can make these at least as large as the number of hosts in the cluster plus 10, assuming that any MoM only has one `scp` session open at any one time.

12.9.5.5.i Alternatives to Changing `ssh` Limits

To avoid having to change limits on incoming `ssh` connections, you can do the following:

- Use a mounted directory and employ `usecp` MoM parameters. See section 12.9.5.2, “Avoiding Remote Transfers in Large Complexes”, on page 887.
- Use compression to service more requests with the same amount of hardware resources. See section 12.9.5.3, “Improving Performance for `ssh`”, on page 887.

12.9.5.6 Getting Around Bandwidth Limits

If you have bandwidth limits, you can use a command such as `gsiftp`, which allows you to specify the bandwidth you want to use for file transfer. Follow the instructions in section 12.9.3.3, “Configuring MoM to Use Different Flags, a Script, or a Different Command”, on page 885.

12.9.6 General Advice

12.9.6.1 Using `scp` for Security

Unless your complex is a closed system, we recommend using `scp` instead of `rcp`, because `scp` is more secure.
12.9.6.2 Avoiding Asynchronous Writes to NFS

Asynchronous writes to an NFS server can cause reliability problems. If using an NFS file system, mount the NFS file system synchronously (without caching.)

12.9.6.3 Returning Output on Cray

If your site has disabled the use of remote operation functions ("r" commands) and output cannot be returned for jobs running on compute nodes, enable the use of the cp command by adding $usecp to the PBS_HOME/mom_priv/config file on each login node. See section 12.9.2, “Configuring MoM for Local Copy”, on page 883.

12.9.6.4 Editing the pbs.conf File Under Windows

You can edit the pbs.conf file by calling the PBS program named “pbs-config-add”. For example, on 32-bit Windows systems:

\Program Files\PBS Pro\exec\bin\pbs-config-add “PBS_SCP=\winnt\scp.exe”

On 64-bit Windows systems:

\Program Files (x86)\PBS Pro\exec\bin\pbs-config-add

“PBS_SCP=\winnt\scp.exe”

Do not edit pbs.conf directly; this could reset the permission on the file, which could prevent other users from running PBS.

12.9.6.5 The pbs_rcp Command

12.9.6.5.i Exit Values for pbs_rcp

The pbs_rcp command exits with a non-zero exit status for any error. This tells MoM whether or not the file was delivered.

12.9.6.5.ii The pbs_rshd Windows Service

The Windows version of PBS contains a service called pbs_rshd for supporting remote file copy requests for delivering job output and error files to destination hosts. The pbs_rshd service supports pbs_rcp, but does not allow normal rsh activities. This service is used only by PBS; it is not intended to be used by administrators or users. See “The pbs_rshd Windows Service” on page 3 in the PBS Professional Installation & Upgrade Guide.

12.9.7 Caveats

- Output is not delivered if the path specified by PBS_SCP or PBS_RCP in pbs.conf
is incorrect.

- When a job is rerun, its stdout and stderr files are sent to the server and stored in PBS_HOME/spool. When the job is sent out for execution again, its stdout and stderr are sent with it. The copy mechanism used for these file transfers is internal to PBS; you cannot alter it or manage it in any way.

12.10 Temporary File Location for PBS Components

You can configure where all PBS components put their temporary files and directories on each system. You may want to avoid using the usual temporary file locations of /tmp and /var/tmp, because users tend to fill these up.

12.10.1 Default Location for Temporary Files

By default, on UNIX/Linux platforms, PBS components put their temporary files and directories in /var/tmp. PBS uses this location because it is persistent across restarts or crashes, allowing diagnosis of a problem, whereas the contents of /tmp may be lost.

On Windows, the default location is C:\WINNT\TEMP if it is present, or C:\WIN-DOWS\TEMP.

12.10.2 Configuring Temporary File Location for PBS Components

You configure the location of temporary files and directories for PBS components by setting the value of the PBS_TMPDIR configuration parameter in the /etc/pbs.conf file on each system. Set this parameter to the directory to be used for storing temporary files and directories by all PBS components on that system.

After you set the location of temporary files and directories, restart all PBS components:

<path to init.d>/init.d/pbs restart
The location for temporary files and directories for PBS components is determined by the following settings, in order of decreasing precedence:

1. $tmpdir in mom_priv/config (affects pbs_mom only, not other components)
2. PBS_TMPDIR environment variable
3. TMPDIR (for UNIX/Linux) or TMP (for Windows) environment variable
4. PBS_TMPDIR in PBS configuration file
5. If none of the preceding settings are present, PBS uses default values:
   - /var/tmp (for UNIX/Linux)
   - C:\WINNT\TEMP or C:\WINDOWS\TEMP (for Windows)

### 12.10.3 Requirements

- The specified directory must exist.
  
  If the configured temporary file location does not exist, PBS prints the following error message:
  
  `<command>`: No such file or directory (2) in chk_file_sec, Security violation "<directory>" resolves to "<directory>"

  `<command>`: Unable to configure temporary directory.

- The directory must be globally readable and writable.

- On UNIX/Linux systems, the directory must have the sticky bit set in the file permissions.

- The directory must not present a security risk:
  
  - All parent directories of the configured temporary directory must be owned by a UID less than 11 and a GID less than 10.
    
    - If the assigned owner has write permission, the UID must be 10 or less.
    
    - If the assigned group has write permission, the GID must be 9 or less.
  
  - Each parent directory must not be writable by "other".

  If a PBS component detects a security risk for a file or directory, it prints the following messages and exits:

  `<command>`: Not owner (1) in chk_file_sec, Security violation "<directory>" resolves to "<directory>"

  `<command>`: Unable to configure temporary directory.
12.10.4 Advice and Recommendations for Temporary File Location

- Make sure that the location you choose for temporary files is cleaned periodically.
- In the past, some PBS components defaulted to `/tmp` for storing temporary files. All components now default to `/var/tmp`, which is most likely a persistent storage location. You should take this into account and adjust the cleaning of `/var/tmp` accordingly.
- If a PBS component prints a security error message and exits, fix the security problem and restart the component.

12.11 Administration Caveats

12.11.1 Windows Caveats

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.
Chapter 13

Problem Solving

The following is a list of common problems and recommended solutions. Additional information is always available online at the PBS website, www.example.com/UserArea. The last section in this chapter gives important information on how to get additional assistance from the PBS Support staff.

13.1 Debugging PBS

To set the size of the core file for a PBS daemon, you can set PBS_CORE_LIMIT in pbs.conf. Set this on the machine where the daemon runs. This can be set to an integer number of bytes or to the string "unlimited".

13.2 Server Host Bogs Down After Startup

If the server host becomes unresponsive a short time after startup, the server may be trying to contact the wrong license server.

13.2.1 Symptoms

15 seconds to one or two minutes after you start the PBS server, the system becomes unresponsive.
### 13.2.2 Problem

The problem may be caused by the `pbs_license_info` server attribute pointing to an old FLEX license server. This attribute should point to the new LM-X license server. See “Configuring PBS for Licensing” on page 93 in the PBS Professional Installation & Upgrade Guide.

### 13.2.3 Treatment

On some Linux systems, the effects of memory starvation on subsequent responsiveness may be long-lasting. Therefore, instead of merely killing and restarting the PBS server, we recommend rebooting the machine.

Take the following steps:

1. Reboot the machine into single-user mode.
2. Determine the correct value for `pbs_license_info` and set the `PBS_LICENSE_FILE_LOCATION` entry in `pbs.conf` to this value.
3. Reboot, or change runlevel to multi-user.
4. Using `qmgr`, set the `pbs_license_info` server attribute to the correct value:
   ```
   # qmgr -c "set server pbs_license_info = <value of PBS_LICENSE_FILE_LOCATION entry in pbs.conf>"
   ```
5. Stop the PBS server process.
6. Remove the `PBS_LICENSE_FILE_LOCATION` entry in `pbs.conf`.
7. Continue normally.

### 13.3 Finding PBS Version Information

Use the `qstat` command to find out what version of PBS Professional you have.

```qstat -fB
```

In addition, each PBS command will print its version information if given the `--version` option. This option cannot be used with other options.

### 13.4 Troubleshooting and Hooks

You may wish to disable hook execution in order to debug PBS issues. To verify whether hooks are part of the problem, disable each hook by setting its `enabled` attribute to `False`.
13.5 Directory Permission Problems

If for some reason the access permissions on the PBS file tree are changed from their default settings, a component of the PBS system may detect this as a security violation, and refuse to execute. If this is the case, an error message to this effect will be written to the corresponding log file. You can run the `pbs_probe` command to check (and optionally correct) any directory permission (or ownership) problems. See “pbs_probe” on page 70 of the PBS Professional Reference Guide for details on usage of the `pbs_probe` command.

13.6 Common Errors

13.6.1 Clients Unable to Contact Server

If a client command (such as `qstat` or `qmgr`) is unable to connect to a Server there are several possibilities to check. If the error return is 15034, “No server to connect to”, check (1) that there is indeed a Server running and (2) that the default Server information is set correctly. The client commands will attempt to connect to the Server specified on the command line if given, or if not given, the Server specified by `SERVER_NAME` in `pbs.conf`.

If the error return is 15007, “No permission”, check for (2) as above. Also check that the executable `pbs_iff` is located in the search path for the client and that it is setuid root. Additionally, try running `pbs_iff` by typing:

```
pbs_iff -t server_host 15001
```

Where `server_host` is the name of the host on which the Server is running and 15001 is the port to which the Server is listening (if started with a different port number, use that number instead of 15001). Check for an error message and/or a non-zero exit status. If `pbs_iff` exits with a non-zero status, either the Server is not running or was installed with a different encryption system than was `pbs_iff`.

13.6.2 Vnodes Down

The PBS Server determines the state of vnodes (up or down), by communicating with MoM on the vnode. The state of vnodes may be listed by two commands: `qmgr` and `pbsnodes`

```
Qmgr: list node @active
pbsnodes -a
    Node jupiter  state = state-unknown, down
```
A vnode in PBS may be marked “down” in one of two substates. For example, the state above of vnode “jupiter” shows that the Server has not had contact with MoM since the Server came up. Check to see if a MoM is running on the vnode. If there is a MoM and if the MoM was just started, the Server may have attempted to poll her before she was up. The Server should see her during the next polling cycle in 10 minutes. If the vnode is still marked “state-unknown, down” after 10+ minutes, either the vnode name specified in the Server’s node file does not map to the real network hostname or there is a network problem between the Server’s host and the vnode.

If the vnode is listed as:

```
pbsnodes -a
Node jupiter state = down
```

then the Server has been able to ping MoM on the vnode in the past, but she has not responded recently. The Server will send a “ping” PBS message to every free vnode each ping cycle, 10 minutes. If a vnode does not acknowledge the ping before the next cycle, the Server will mark the vnode down.

### 13.6.3 Requeueing a Job “Stuck” on a Down Vnode

PBS Professional will detect if a vnode fails when a job is running on it, and will automatically requeue and schedule the job to run elsewhere. If the user marked the job as “not rerunnable” (i.e. via the `qsub -r n` option), then the job will be deleted rather than requeued. If the affected vnode is vnode 0 (Mother Superior), the requeue will occur quickly. If it is another vnode in the set assigned to the job, it could take a few minutes before PBS takes action to requeue or delete the job. However, if the auto-requeue feature is not enabled, or if you wish to act immediately, you can manually force the requeueing and/or rerunning of the job. See section 9.4.2, “Node Fail Requeue: Jobs on Failed Vnodes”, on page 734.

If you wish to have PBS simply remove the job from the system, use the “–Wforce” option to `qdel`:

```
qdel -Wforce jobID
```

If instead you want PBS to requeue the job, and have it immediately eligible to run again, use the “–Wforce” option to `qrerun`

```
qrerun -Wforce jobID
```

See "Job Input and Output Files", on page 49 of the PBS Professional User’s Guide.
13.6.4 Job Cannot be Executed

If a user receives a mail message containing a job ID and the line “Job cannot be executed”, the job was aborted by MoM when she tried to place it into execution. The complete reason can be found in one of two places, MoM’s log file or the standard error file of the user’s job. If the second line of the message is “See Administrator for help”, then MoM aborted the job before the job’s files were set up. The reason will be noted in MoM’s log. Typical reasons are a bad user/group account, checkpoint/restart file (Cray or SGI), or a system error. If the second line of the message is “See job standard error file”, then MoM had created the job’s file and additional messages were written to standard error. This is typically the result of a bad resource request.

13.6.5 Running Jobs with No Active Processes

On very rare occasions, PBS may be in a situation where a job is in the Running state but has no active processes. This should never happen as the death of the job’s shell should trigger MoM to notify the Server that the job exited and end-of-job processing should begin. If this situation is noted, PBS offers a way out. Use the qsig command to send SIGNULL, signal 0, to the job. (Usage of the qsig command is provided in the PBS Professional User’s Guide.) If MoM finds there are no processes then she will force the job into the exiting state.

13.6.6 Job Held Due to Invalid Password

If a job fails to run due to an invalid password, then the job will be put on hold (hold type “p”), its comment field updated as to why it failed, and an email sent to user for remedy action. See also the qhold and qrls commands in the PBS Professional User’s Guide.

13.6.7 SuSE 9.1 with mpirun and ssh

Use “ssh -n” instead of “ssh”.

13.6.8 Jobs that Can Never Run

If backfilling is being used, the scheduler looks at the job being backfilled around and determines whether that job can never run.

If backfilling is turned on, the scheduler determines whether that job can or cannot run now, and if it can't run now, whether it can ever run. If the job can never run, the scheduler logs a message saying so.
Chapter 13  Problem Solving

The scheduler only considers the job being backfilled around. That is the only job for which it will log a message saying the job can never run.

This means that a job that can never run will sit in the queue until it becomes the most deserving job. Whenever this job is considered for having small jobs backfilled around it, the error message “resource request is impossible to solve: job will never run” is printed in the scheduler's log file. If backfilling is off, this message will not appear.

If backfilling is turned off, the scheduler determines only whether that job can or cannot run now. The scheduler won’t determine if a job will ever run or not.

13.6.9  Server Won’t Start

- The Server may not start due to problems with the data service. Call PBS technical support; see “Technical Support” on page ii. For more on the PBS data service, see “pbs_dataservice” on page 46 of the PBS Professional Reference Guide.

13.6.10  PBS Data Service Doesn’t Start

- You may need to create the data service user account. This must be creating before installing PBS. See “Create Required Accounts” on page 41 in the PBS Professional Installation & Upgrade Guide.

13.6.11  Problem With Dynamic Resource

If you need to debug a dynamic resource being supplied by an external script, it may help to follow these steps:

1. Set the scheduler’s log_filter parameter to zero (everything is logged)
   - Edit PBS_HOME/sched_priv/sched_config
   - Change the log_filter line to 0
2. Send a SIGHUP to the scheduler (pbs_sched)
3. The scheduler log will contain the value the scheduler reads from the external script

13.6.12  Can’t Create Formula or Hook

You must run qmgr at the server host when operating on the server’s job_sort_formula attribute or on hooks. For example, attempting to create the formula at another host will result in the following error:

```
qmgr obj= svr=default: Unauthorized Request job_sort_formula
```
13.6.13 Data Service Running When PBS Server is Down

You can use the `pbs.dataservice` command to stop the data service. See “`pbs.dataservice`” on page 46 of the PBS Professional Reference Guide.

13.6.14 Unrecognized Timezone Variable

In order to create reservations, the PBS server must recognize the `PBS_TZID` environment variable at the submission host. The appropriate zone location for the submission host can be obtained from the machine on which the PBS Professional server is installed.

- On Linux platforms, either use the `tzselect` command, if it is available, or look in the underlying operating system's `zone.tab` timezone location file, which may be found under `/usr/share/zoneinfo/zone.tab`.
- On all other platforms, look in the list of libical supported zoneinfo locations available under `$PBS_EXEC/lib/ical/zoneinfo/zones.tab`.

13.6.15 Bad Vnode on Startup

If, when the server starts up, one or more vnodes cannot be resolved, the server marks the bad vnode(s) in state “`state-unknown, down`”.

13.7 Errors on Windows

This section discusses errors encountered under Windows.

13.7.1 Windows: Services Don’t Start

In the case where the PBS daemons, the Active Directory database, and the domain controller are all on the same host, some PBS services may not start up immediately. If the Active Directory services are not running when the PBS daemons are started, the daemons won’t be able to talk to the domain controller. This can prevent the PBS daemons from starting. As a workaround, wait until the host is completely up, then retry starting the failing service.

Example:

```
net start pbs_server
```
13.7.2 MoMs Won’t Start

In a domained environment, if the PBS service account is a member of any group besides “Domain Users”, the install program will fail to add the PBS service account to the local Administrators group on the install host. Make sure that the PBS service account is a member of only one group, “Domain Users” in a domained environment.

13.7.3 Windows: qstat Errors

If the qstat command produces an error such as:

```
  illegally formed job identifier.
```

This means that the DNS lookup is not working properly, or reverse lookup is failing. Use the following command to verify DNS reverse lookup is working:

```
pbs_hostn -v hostname
```

If however, qstat reports “No Permission”, then check pbs.conf, and look for the entry “PBS_EXEC”. qstat (in fact all the PBS commands) will execute the command “PBS_EXEC\sbin\pbs iff” to do its authentication. Ensure that the path specified in pbs.conf is correct.

13.7.4 Windows: qsub Errors

If, when attempting to submit a job to a remote server, qsub reports:

```
  BAD uid for job execution
```

Then you need to add an entry in the remote system's .rhosts or hosts.equiv pointing to your Windows machine. Be sure to put in all hostnames that resolve to your machine. See section 8.9.3, “User Authentication”, on page 676.

If remote account maps to an Administrator-type account, then you need to set up a .rhosts entry, and the remote server must carry the account on its acl_roots list.

13.7.5 Windows: Server Reports Error 10035

If Server is not able to contact the Scheduler running on the same local host, it may print to its log file the error message,

```
  10035 (Resources Temporarily Unavailable)
```
This is often caused by the local hostname resolving to a bad IP address. Perhaps, in \%WINDIR\%\system32\drivers\etc\hosts, localhost and hostname were mapped to 127.0.0.1.

13.7.6 Windows: Server Reports Error 10054

If the Server reports error 10054 \texttt{rp\_request()}, this indicates that another process, probably \texttt{pbs\_sched}, \texttt{pbs\_mom}, or \texttt{pbs\_send\_job} is hung up causing the Server to report bad connections. If you desire to kill these services, then use Task Manager to find the Service’s process ID, and then issue the command:

\begin{verbatim}
pbskill process-id
\end{verbatim}

13.7.7 Windows: PBS Permission Errors

If the Server, MoM, or Scheduler fails to start up because of permission problems on some of its configuration files like \texttt{pbs\_environment} or \texttt{mom\_priv/config}, then correct the permission by running:

\begin{verbatim}
pbs_mkdirs server
pbs_mkdirs mom
pbs_mkdirs sched
\end{verbatim}

13.7.8 Windows: Errors When Not Using Drive C:

If PBS is installed on a hard drive other than C:, it may not be able to locate the \texttt{pbs.conf} global configuration file. If this is the case, PBS will report the following message:

\begin{verbatim}
E:\Program Files\PBS Pro\exec\bin>qstat -
pbsconf error: pbs conf variables not found:
PBS\_HOME PBS\_EXEC
No such file or directory
qstat: cannot connect to server UNKNOWN (errno=0)
\end{verbatim}

To correct this problem, set \texttt{PBS\_CONF} to point \texttt{pbs.conf} to the right path. Normally, during PBS Windows installation, this would be set in system \texttt{autoexec.bat} which will be read after the Windows system has been restarted. Thus, after PBS Windows installation completes, be sure to reboot the Windows system in order for this variable to be read correctly.
13.7.9 Windows: Vnode Comment “ping: no stream”

If a vnode shows a “down” status in xpbsmon or “pbsnodes -a” and contains a vnode comment with the text “ping: no stream” and “write err”, then attempt to restart the Server as follows to clear the error:

```
net stop pbs_server
net start pbs_server
```

13.7.10 Windows: Services Debugging Enabled

The PBS services, `pbs_server`, `pbs_mom`, `pbs_sched`, and `pbs_rshd` are compiled with debugging information enabled. Therefore you can use a debugging tool (such as Dr. Watson) to capture a crash dump log which will aid the developers in troubleshooting the problem. To configure and run Dr. Watson, execute `drwtsn32` on the Windows command line, set its “Log Path” appropriately and click on the button that enables a popup window when Dr. Watson encounters an error. Then run a test that will cause one of the PBS services to crash and email to PBS support the generated output in Log_Path. Other debugging tools may be used as well.

13.7.11 Windows: Client Commands Slow

PBS caches the IP address of the local host, and uses this to communicate between the server, scheduler, and MoM. If the cached IP address is invalidated, PBS can become slow. In both scenarios, jobs must be killed and restarted.

13.7.11.1 Scenario 1: Wireless Router, DHCP Enabled

The system is connected to a wireless router that has DHCP enabled. DHCP returned a new IP address for the server short name, but DNS is resolving the server full name to a different IP address.

The IP address and server full name have become invalid due to the new DHCP address. PBS has cached the IP address of the server full name.

Therefore, the PBS server times out when trying to connect to the scheduler and local MoM using the previously cached IP address. This makes PBS slow.
Problem Solving

Chapter 13

Symptom:
1. PBS is slow.
   a. Server logs show "Could not contact scheduler".
   b. `pbsnodes -a` shows that the local node is down.
2. First IP addresses returned below don't match:
   ```
   cmd.admin> pbs_hostn -v <server_short_name>
   cmd.admin> pbs_hostn -v <server_full_name>
   ```
Workaround: cache the correct new IP address of the local server host.
1. Add the address returned by `pbs_hostn -v <server_short_name>` (normally the DHCP address) to `%WINDIR%\system32\drivers\etc\hosts` file as follows:
   ```
   <DHCP address> <server_full_name> <server_short_name>
   ```
2. Restart all the PBS services:
   ```
   cmd.admin> net stop pbs_sched
   cmd.admin> net stop pbs_mom
   cmd.admin> net stop pbs_rshd
   cmd.admin> net stop pbs_server
   ```
   ```
   cmd.admin> net start pbs_sched
   cmd.admin> net start pbs_mom
   cmd.admin> net start pbs_rshd
   cmd.admin> net start pbs_server
   ```

### 13.7.11.2 Scenario 2: DHCP-Enabled Environment

The system is running in a DHCP-enabled environment. Both the server short name and server full name resolve to the same DHCP address. Then the DHCP address expires and the local server host gets a new address, invalidating what's been cached by PBS.
Problem Solving

Symptom:

1. PBS is slow.
   a. Server logs show "Could not contact scheduler".
   b. `pbsnodes -a` shows local node is down.

2. The first IP addresses below match, but it's now a different IP address:

   ```
   cmd.admin> pbs_hostn -v <server_short_name>
   cmd.admin> pbs_hostn -v <server_full_name>
   ```

   Workaround: obtain the correct new IP address of the local server host.

1. Simply restart all the PBS services:

   ```
   cmd.admin> net stop pbs_sched
   cmd.admin> net stop pbs_mom
   cmd.admin> net stop pbs_rshd
   cmd.admin> net stop pbs_server
   ```

   ```
   cmd.admin> net start pbs_sched
   cmd.admin> net start pbs_mom
   cmd.admin> net start pbs_rshd
   cmd.admin> net start pbs_server
   ```

   **IMPORTANT:**
   
   When contacting PBS Professional Support, please provide as much of the following information as possible:
   
   • PBS SiteID
   • Output of the following commands:
     `qstat -Bf`
     `qstat -Qf`
     `pbsnodes -a`
   • If the question pertains to a certain type of job, include:
     `qstat -f job_id`
   • If the question is about scheduling, also send your `(PBS_HOME)/sched_priv/sched_config` file.

To expand, renew, or change your PBS support contract, contact our Sales Department. (See contact information on the inside front cover of this manual.)
13.8 Troubleshooting PBS Licenses

13.8.1 Wrong License Server: Out of Memory

If you run out of memory shortly after startup, the server may be looking for the wrong license server. See section 13.2, “Server Host Bogs Down After Startup”, on page 893.

13.8.2 Unable to Connect to License Server

If PBS cannot contact the license server, the server will log a message:

"Unable to connect to license server at pbs_license_info=<X>"

If the license file location is incorrectly initialized (e.g. if the host name or port number is incorrect), PBS may not be able to pinpoint the misconfiguration as the cause of the failure to reach a license server.

If PBS cannot detect a license server host and port when it starts up, the server logs an error message:

"Did not find a license server host and port (pbs_license_info=<X>). No external license server will be contacted"

13.8.3 Unable to Run Job; Unable to Obtain Licenses

If the PBS scheduler cannot obtain the licenses to run or resume a job, the scheduler will log a message:

"Could not run job <job>; unable to obtain <N> CPU licenses. avail licenses=<Y>"

"Could not resume <job>; unable to obtain <N> CPU licenses. avail licenses=<Y>"

13.8.4 Job in Reservation Fails to Run

A job in a reservation may not be able to run due to a shortage of licenses. The scheduler will log a message similar to the following:

"Could not run job <job>; unable to obtain <N> CPU licenses. avail licenses=<Y>"
If the value of the `pbs_license_min` attribute is less than the number of CPUs in the PBS complex when a reservation is being confirmed, the server will log a warning:

"WARNING: reservation <resID> confirmed, but if reservation starts now, its jobs are not guaranteed to run as pbs_license_min=<X> < <Y> (# of CPUs in the complex)"

### 13.8.5 New Jobs Not Running

If PBS loses contact with the Altair License Server, any jobs currently running will not be interrupted or killed. The PBS server will continually attempt to reconnect to the license server, and re-license the assigned vnodes once the contact to the license server is restored. No new jobs will run if PBS server loses contact with the License server.

### 13.8.6 Insufficient Minimum Licenses

If the PBS server cannot get the number of licenses specified in `pbs_license_min` from the license server, the server will log a message:

"checked-out only <X> CPU licenses instead of pbs_license_min=<Y> from license server at host <H>, port <P>. Will try to get more later."

### 13.8.7 Wrong Type of License

If the PBS server encounters a proprietary license key that is of not type “T”, then the server will log the following message:

"license key #1 is invalid: invalid type or version."

### 13.8.8 User Error Messages

If a user's job could not be run due to unavailable licenses, the job will get a comment:

"Could not run job <job>; unable to obtain <N> CPU licenses. avail_licenses=<Y>"

If a user's job could not be resumed due to unavailable licenses, the job will get a comment:

"Could not resume job <job>; unable to obtain <N> CPU licenses. avail_licenses=<Y>"
13.9 Security-related Problems

13.9.1 PBS Daemon Will Not Start

If the PBS Server, MoM, or Scheduler fails to start up, it may be refusing to start because it has detected permissions problems in its directories or on one or more of its configuration files, such as pbs_environment or mom_priv/config.

13.9.1.1 Correcting Permissions Problems on UNIX/Linux

You can use the pbs_probe command to detect and repair file and directory permissions problems. You can run pbs_probe in report mode or fix mode; in report mode, it reports the errors found; in fix mode, it attempts to fix detected problems, and reports any problems it could not fix.

To fix permissions errors, log into the host you wish to check, and run the following command:

```
pbs_probe -f
```

See the pbs_probe(8B) manual page.

13.9.1.2 Correcting Permissions Problems on Windows

You can use the pbs_mkdirs command to correct file and directory permissions problems on Windows. The command checks and if necessary repairs the permissions of configuration files such as pbs_environment and mom_priv/config. You should run the pbs_mkdirs command only while the PBS services are stopped.

To repair permissions on the Server/Scheduler host, log into the host and run the following commands:

```
net stop pbs_server
net stop pbs_sched
pbs_mkdirs server
pbs_mkdirs sched
net start pbs_server
net start pbs_sched
```
Chapter 13  Problem Solving

To repair permissions on an execution host, log into the host and run the following commands:

```
net stop pbs_mom
pbs_mkdirs mom
net start pbs_mom
```

13.9.2  Crash Recovery

PBS daemons could terminate unexpectedly either because the host machine stops running or because the daemon itself stops running. The daemon may be killed by mistake, or may (rarely) crash. The Server may terminate if the filesystem runs out of space.

13.9.2.1  Recovery When Host Machine Stops

If the host machine stops running, no special steps are required, since PBS will be started when the machine starts.

13.9.2.1.i  Execution Host Stops

If the host machine is an execution host, any jobs that were running on that host were terminated when the machine stopped, and when MoM is restarted, she will report to the Server that those jobs are dead, and begin normal activity. The Server will automatically restart any jobs that can be restarted.

Shutting down one host of a multi-host job will cause that job to be killed. The job will have to be rerun; restarting the MoM on the stopped host with the `-p` option will not help the job. See section 13.9.2.2.ii, “MoM Start Options”, on page 909.

13.9.2.1.ii  Server/Scheduler Host Stops

If the host machine is the Server/Scheduler host, no data is lost and no jobs are lost, because the Server writes everything to disk. The Server is restarted automatically upon machine startup.

The Scheduler is started automatically upon machine startup. The Scheduler starts fresh each cycle, so it does not lose data.

13.9.2.2  Recovery When Daemon Stops

For more detailed information on starting and stopping PBS, see “Starting and Stopping PBS” on page 177 in the PBS Professional Installation & Upgrade Guide.
13.9.2.2.i  Restarting MoM

If the daemon that stops running is the MoM, you can restart it according to your needs. The default is for MoM to start up as if the machine had just been started. You can choose to preserve running jobs, to kill and requeue running jobs, or to have MoM report her previous jobs to the Server as dead.

The command that starts the PBS MoM is PBS_EXEC/sbin/pbs_mom.

13.9.2.2.ii  MoM Start Options

(No options)
MoM does not kill any processes. She reports to the Server that her previous jobs are dead.

-r
MoM kills all her jobs. The Server requeues any jobs that can be rerun.

-p
MoM leaves any running jobs in the running state.

Be careful when starting MoM with the -p or -r options on a newly-booted system, because these options depend on the assumption that the PIDs from the previous invocation are still valid.

13.9.2.2.iii  Restarting the Server

If the daemon that stops running is the Server, you can restart it according to your needs. The default is for the Server to start up, leaving jobs in their pre-shutdown state. You can choose to preserve running jobs, to purge all jobs, or to re-create the PBS complex (not recommended). In most cases, you should allow the Server to be restarted by the PBS start script.

13.9.2.2.iv  Server Start Options

(No options) or
-t warm
The Server leaves all jobs in their pre-crash states. The Server starts a normal scheduling cycle, without giving special precedence to jobs that were killed and requeued upon shutdown.

-t hot
The Server leaves all running jobs running. The Server attempts to start any jobs that were killed and requeued upon shutdown before starting any new jobs.

-t cold
The Server purges all jobs, but retains its configuration information.
Chapter 13  Problem Solving

- t create
  Not recommended. The Server starts everything fresh. It purges all jobs, and deletes all information about the PBS complex, including attribute settings and queues.

13.9.2.2.v  Restarting the Scheduler
If the Scheduler was killed, you must restart it using this command:

  pbs_sched

If the Scheduler crashed, it will restart itself. If the Scheduler crashed due to a SIGSEGV or a SIGBUS, it will wait until five minutes have passed before restarting itself. You can restart by hand it at any time.

13.9.2.3  Filesystem Runs Out of Space
If your filesystem has run out of space, the Server may experience errors or may crash. If the Server is still running, you need only to free up enough space. If the Server has crashed, you must restart it. See section 13.9.2.2.iii, “Restarting the Server”, on page 909.

13.10  Time Zone Problems
Problem: you see this message:

  pbs_rsub: Bad time specification(s)

Reason: The time zone is not specified correctly in PBS_TZID. On later Linux updates, the system's zoneinfo files may have some countries represented under different names from those in previous releases. For example, Asia/Calcutta has been replaced by Asia/Kolkata.

Workaround: While the PBS server is running and can contact the execution machine, use the Linux tzselect utility to determine the value for PBS_TZID.

13.11  Job Comments for Problem Jobs
PBS can detect when a job cannot run with the current unused resources and when a job will never be able to run with all of the configured resources. PBS can set the job’s comment attribute to reflect why the job is not running.
Problem Solving

If the job’s comment starts with “Can never run”, the job will never be able to run with the resources that are currently configured. This can happen when:

- A job requests more of a consumable resource than is available on the entire complex
- A job requests a non-consumable resource that is not available on the complex

For example, if there are 128 total CPUs in the complex, and the job requests 256 CPUs, the job’s comment will start with this message.

If the job’s comment starts with “Not running”, the job cannot run with the resources that are currently available. For example, if a job requests 8 CPUs and the complex has 16 CPUs but 12 are in use, the job’s comment will start with this message.

You may see the following comments. R is for “Requested”, A is for “Available”, and T is for “Total”:

- “Not enough free nodes available”
- “Not enough total nodes available”
- “Job will never run with the resources currently configured in the complex”
- “Insufficient amount of server resource <resource> (R | A | T | <requested value> !=<available values for requested resource>)”
- “Insufficient amount of queue resource <resource> (R | A | T | <requested value> !=<available values for requested resource>)”
- “Error in calculation of start time of top job”
- “Can’t find start time estimate”

The “Can Never Run” prefix may be seen with the following messages:

- “Insufficient amount of resource <resource> (R | A | T | <requested value> !=<available values for requested resource>)”
- “Insufficient amount of Server resource <resource> (R | A | T | <requested value> !=<available values for requested resource>)”
- “Insufficient amount of Queue resource <resource> (R | A | T | <requested value> !=<available values for requested resource>)”
- “Not enough total nodes available”
- “can’t fit in the largest placement set, and can’t span psets”

13.12 Getting Help

If the material in the PBS manuals is unable to help you solve a particular problem, you may need to contact the PBS Support Team for assistance. First, be sure to check the Customer Login area of the PBS Professional website, which has a number of ways to assist you in resolving problems with PBS, such as the Tips & Advice page.
Chapter 13  

**Problem Solving**

The PBS Professional support team can also be reached directly via email and phone (contact information on the inside front cover of this manual).
Appendix A: License Agreement

CAUTION!
PRIOR TO INSTALLATION OR USE OF THE SOFTWARE YOU MUST CONSENT TO THE FOLLOWING SOFTWARE LICENSE TERMS AND CONDITIONS BY CLICKING THE “I ACCEPT” BUTTON BELOW. YOUR ACCEPTANCE CREATES A BINDING LEGAL AGREEMENT BETWEEN YOU AND ALTAIR. IF YOU DO NOT HAVE THE AUTHORITY TO BIND YOUR ORGANIZATION TO THESE TERMS AND CONDITIONS, YOU MUST CLICK “I DO NOT ACCEPT” AND THEN HAVE AN AUTHORIZED PARTY IN THE ORGANIZATION THAT YOU REPRESENT ACCEPT THESE TERMS.

IF YOU, OR THE ORGANIZATION THAT YOU REPRESENT, HAS A MASTER SOFTWARE LICENSE AGREEMENT (“MASTER SLA”) ON FILE AT THE CORPORATE HEADQUARTERS OF ALTAIR ENGINEERING, INC. (“ALTAIR”), THE MASTER SLA TAKES PRECEDENCE OVER THESE TERMS AND SHALL GOVERN YOUR USE OF THE SOFTWARE.

MODIFICATION(S) OF THESE SOFTWARE LICENSE TERMS IS EXPRESSLY PROHIBITED. ANY ATTEMPTED MODIFICATION(S) WILL BE NONBINDING AND OF NO FORCE OR EFFECT UNLESS EXPRESSLY AGREED TO IN WRITING BY AN AUTHORIZED CORPORATE OFFICER OF ALTAIR. ANY DISPUTE RELATING TO THE VALIDITY OF AN ALLEGED MODIFICATION SHALL BE DETERMINED IN ALTAIR’S SOLE DISCRETION.
Appendix A: License Agreement

Altair Engineering, Inc. - Software License Agreement

THIS SOFTWARE LICENSE AGREEMENT, including any Additional Terms (together with the "Agreement"), shall be effective as of the date of YOUR acceptance of these software license terms and conditions (the "Effective Date") and is between ALTAIR ENGINEERING, INC., 1820 E. Big Beaver Road, Troy, MI 48083-2031, USA, a Michigan corporation ("Altair"), and YOU, or the organization on whose behalf you have authority to accept these terms (the "Licensee"). Altair and Licensee, intending to be legally bound, hereby agree as follows:

1. DEFINITIONS. In addition to terms defined elsewhere in this Agreement, the following terms shall have the meanings defined below for purposes of this Agreement:

Additional Terms. Additional Terms are those terms and conditions which are determined by an Altair Subsidiary to meet local market conditions.

Documentation. Documentation provided by Altair or its resellers on any media for use with the Products.

Execute. To load Software into a computer's RAM or other primary memory for execution by the computer.

Global Zone: Software is licensed based on three Global Zones: the Americas, Europe and Asia-Pacific. When Licensee has Licensed Workstations located in multiple Global Zones, which are connected to a single License (Network) Server, a premium is applied to the standard Software License pricing for a single Global Zone.

ISV/Independent Software Vendor. A software company providing its products, ("ISV Software") to Altair's Licensees through the Altair License Management System using Altair License Units.

License Log File. A computer file providing usage information on the Software as gathered by the Software.

License Management System. The license management system (LMS) that accompanies the Software and limits its use in accordance with this Agreement, and which includes a License Log File.

License (Network) Server. A network file server that Licensee owns or leases located on Licensee's premises and identified by machine serial number and/or HostID on the Order Form.

License Units. A parameter used by the LMS to determine usage of the Software permitted under this Agreement at any one time.

Licensed Workstations. Single-user computers located in the same Global Zone(s) that Licensee owns or leases that are connected to the License (Network) Server via local area network or Licensee's private wide-area network.
Appendix A: License Agreement

Maintenance Release. Any release of the Products made generally available by Altair to its Licensees with annual leases, or those with perpetual licenses who have an active maintenance agreement in effect, that corrects programming errors or makes other minor changes to the Software. The fees for maintenance and support services are included in the annual license fee but perpetual licenses require a separate fee.

Order Form. Altair's standard form in either hard copy or electronic format that contains the specific parameters (such as identifying Licensee's contracting office, License Fees, Software, Support, and License (Network) Servers) of the transaction governed by this Agreement.

Products. Products include Altair Software, ISV Software, and/or Suppliers' software; and Documentation related to all of the forgoing.

Proprietary Rights Notices. Patent, copyright, trademark or other proprietary rights notices applied to the Products, packaging or media.

Software. The Altair software identified in the Order Form and any Updates or Maintenance Releases.

Subsidiary. Subsidiary means any partnership, joint venture, corporation or other form of enterprise in which a party possesses, directly or indirectly, an ownership interest of fifty percent (50%) or greater, or managerial or operational control.

Suppliers. Any person, corporation or other legal entity which may provide software or documents which are included in the Software.

Support. The maintenance and support services provided by Altair pursuant to this Agreement.

Templates. Human readable ASCII files containing machine-interpretable commands for use with the Software.

Term. The term of licenses granted under this Agreement. Annual licenses shall have a 12-month term of use unless stated otherwise on the Order Form. Perpetual licenses shall have a term of twenty-five years. Maintenance agreements for perpetual licenses have a 12-month term.

Update. A new version of the Products made generally available by Altair to its Licensees that includes additional features or functionalities but is substantially the same computer code as the existing Products.

2. LICENSE GRANT. Subject to the terms and conditions set forth in this Agreement, Altair hereby grants Licensee, and Licensee hereby accepts, a limited, non-exclusive, non-transferable license to: a) install the Products on the License (Network) Server(s) identified on the Order Form for use only at the sites identified on the Order Form; b) execute the Products on Licensed Workstations in accordance with the LMS for use solely by Licensee's employees, or its onsite Contractors who have agreed to be bound by the terms of this Agreement, for Licensee's internal business use on Licensed Workstations within the Global Zone(s) as iden-
Appendix A: License Agreement

tified on the Order Form and for the term identified on the Order Form; c) make backup copies of the Products, provided that Altair's and its Suppliers' and ISV's Proprietary Rights Notices are reproduced on each such backup copy; d) freely modify and use Templates, and create interfaces to Licensee's proprietary software for internal use only using APIs provided that such modifications shall not be subject to Altair's warranties, indemnities, support or other Altair obligations under this Agreement; and e) copy and distribute Documentation inside Licensee's organization exclusively for use by Licensee's employees and its onsite Contractors who have agreed to be bound by the terms of this Agreement. A copy of the License Log File shall be made available to Altair automatically on no less than a monthly basis. In the event that Licensee uses a third party vendor for information technology (IT) support, the IT company shall be permitted to access the Software only upon its agreement to abide by the terms of this Agreement. Licensee shall indemnify, defend and hold harmless Altair for the actions of its IT vendor(s).

3. RESTRICTIONS ON USE. Notwithstanding the foregoing license grant, Licensee shall not do (or allow others to do) any of the following: a) install, use, copy, modify, merge, or transfer copies of the Products, except as expressly authorized in this Agreement; b) use any back-up copies of the Products for any purpose other than to replace the original copy provided by Altair in the event it is destroyed or damaged; c) disassemble, decompile or “unlock”, reverse translate, reverse engineer, or in any manner decode the Software or ISV Software for any reason; d) sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the Products or Licensee's rights under this Agreement; e) allow use outside the Global Zone(s) or User Sites identified on the Order Form; f) allow third parties to access or use the Products such as through a service bureau, wide area network, Internet location or time-sharing arrangement except as expressly provided in Section 2(b); g) remove any Proprietary Rights Notices from the Products; h) disable or circumvent the LMS provided with the Products; or (i) link any software developed, tested or supported by Licensee or third parties to the Products (except for Licensee's own proprietary software solely for Licensee's internal use).

4. OWNERSHIP AND CONFIDENTIALITY. Licensee acknowledges that all applicable rights in patents, copyrights, trademarks, service marks, and trade secrets embodied in the Products are owned by Altair and/or its Suppliers or ISVs. Licensee further acknowledges that the Products, and all copies thereof, are and shall remain the sole and exclusive property of Altair and/or its Suppliers and ISVs. This Agreement is a license and not a sale of the Products. Altair retains all rights in the Products not expressly granted to Licensee herein. Licensee acknowledges that the Products are confidential and constitute valuable assets and trade secrets of Altair and/or its Suppliers and ISVs. Licensee agrees to take the same precautions necessary to protect and maintain the confidentiality of the Products as it does to protect its own information of a confidential nature but in any event, no less than a reasonable degree of care, and shall not disclose or make them available to any person or entity except as expressly provided in this Agreement. Licensee shall promptly notify Altair in the event any unauthorized person obtains access to the Products. If Licensee is required by any governmental
Appendix A: License Agreement

authority or court of law to disclose Altair's or its ISV's or its Suppliers' confidential information, then Licensee shall immediately notify Altair before making such disclosure so that Altair may seek a protective order or other appropriate relief. Licensee's obligations set forth in Section 3 and Section 4 of this Agreement shall survive termination of this Agreement for any reason. Altair's Suppliers and ISVs, as third party beneficiaries, shall be entitled to enforce the terms of this Agreement directly against Licensee as necessary to protect Supplier's intellectual property or other rights.

Altair and its resellers providing support and training to licensed end users of the Products shall keep confidential all Licensee information provided to Altair in order that Altair may provide Support and training to Licensee. Licensee information shall be used only for the purpose of assisting Licensee in its use of the licensed Products. Altair agrees to take the same precautions necessary to protect and maintain the confidentiality of the Licensee information as it does to protect its own information of a confidential nature but in any event, no less than a reasonable degree of care, and shall not disclose or make them available to any person or entity except as expressly provided in this Agreement.

5. MAINTENANCE AND SUPPORT. Maintenance. Altair will provide Licensee, at no additional charge for annual licenses and for a maintenance fee for paid-up licenses, with Maintenance Releases and Updates of the Products that are generally released by Altair during the term of the licenses granted under this Agreement, except that this shall not apply to any Term or Renewal Term for which full payment has not been received. Altair does not promise that there will be a certain number of Updates (or any Updates) during a particular year. If there is any question or dispute as to whether a particular release is a Maintenance Release, an Update or a new product, the categorization of the release as determined by Altair shall be final. Licensee agrees to install Maintenance Releases and Updates promptly after receipt from Altair. Maintenance Releases and Updates are subject to this Agreement. Altair shall only be obligated to provide support and maintenance for the most current release of the Software and the most recent prior release. Support. Altair will provide support via telephone and email to Licensee at the fees, if any, as listed on the Order Form. If Support has not been procured for any period of time for paid-up licenses, a reinstatement fee shall apply. Support consists of responses to questions from Licensee's personnel related to the use of the then-current and most recent prior release version of the Software. Licensee agrees to provide Altair with sufficient information to resolve technical issues as may be reasonably requested by Altair. Licensee agrees to the best of its abilities to read, comprehend and follow operating instructions and procedures as specified in, but not limited to, Altair's Documentation and other correspondence related to the Software, and to follow procedures and recommendations provided by Altair in an effort to correct problems. Licensee also agrees to notify Altair of a programming error, malfunction and other problems in accordance with Altair's then current problem reporting procedure. If Altair believes that a problem reported by Licensee may not be due to an error in the Software, Altair will so notify Licensee. Questions must be directed to Altair's specially designated telephone support numbers and email addresses. Support will also be available via email at Internet addresses designated by Altair. Support is available
Appendix A: License Agreement

Monday through Friday (excluding holidays) from 8:00 a.m. to 5:00 p.m. local time in the Global Zone where licensed, unless stated otherwise on the Order Form. **Exclusions.** Altair shall have no obligation to maintain or support (a) altered, damaged or Licensee-modified Software, or any portion of the Software incorporated with or into other software not provided by Altair; (b) any version of the Software other than the current version of the Software or the immediately prior release of the Software; (c) problems caused by Licensee's negligence, abuse or misapplication of Software other than as specified in the Documentation, or other causes beyond the reasonable control of Altair; or (d) Software installed on any hardware, operating system version or network environment that is not supported by Altair. Support also excludes configuration of hardware, non-Altair Software, and networking services; consulting services; general solution provider related services; and general computer system maintenance.

6. **WARRANTY AND DISCLAIMER.** Altair warrants for a period of ninety (90) days after Licensee initially receives the Software that the Software will perform under normal use substantially as described in then current Documentation. Supplier software included in the Software and ISV Software provided to Licensee shall be warranted as stated by the Supplier or the ISV. Copies of the Suppliers' and ISV's terms and conditions of warranty are available on the Altair Support website. Support services shall be provided in a workmanlike and professional manner, in accordance with the prevailing standard of care for consulting support engineers at the time and place the services are performed.

**ALTAIR DOES NOT REPRESENT OR WARRANT THAT THE PRODUCTS WILL MEET LICENSEE'S REQUIREMENTS OR THAT THEIR OPERATION WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT IT WILL BE COMPATIBLE WITH ANY PARTICULAR HARDWARE OR SOFTWARE.** **ALTAIR EXCLUDES AND DISCLAIMS ALL EXPRESS AND IMPLIED WARRANTIES NOT STATED HEREIN, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.** **THE ENTIRE RISK FOR THE PERFORMANCE, NON-PERFORMANCE OR RESULTS OBTAINED FROM USE OF THE PRODUCTS RESTS WITH LICENSEE AND NOT ALTAIR.** **ALTAIR MAKES NO WARRANTIES WITH RESPECT TO THE ACCURACY, COMPLETENESS, FUNCTIONALITY, SAFETY, PERFORMANCE, OR ANY OTHER ASPECT OF ANY DESIGN, PROTOTYPE OR FINAL PRODUCT DEVELOPED BY LICENSEE USING THE PRODUCTS.**

7. **INDEMNITY.** Altair will defend and indemnify, at its expense, any claim made against Licensee based on an allegation that the Software infringes a patent or copyright (“Claim”); provided, however, that this indemnification does not include claims which are based on Supplier software or ISV software, and that Licensee has not materially breached the terms of this Agreement, Licensee notifies Altair in writing within ten (10) days after Licensee first learns of the Claim; and Licensee cooperates fully in the defense of the claim. Altair shall have sole control over such defense; provided, however, that it may not enter into any settlement bind-
Appendix A: License Agreement

ing upon Licensee without Licensee's consent, which shall not be unreasonably withheld. If a Claim is made, Altair may modify the Software to avoid the alleged infringement, provided however, that such modifications do not materially diminish the Software's functionality. If such modifications are not commercially reasonable or technically possible, Altair may terminate this Agreement and refund to Licensee the prorated license fee that Licensee paid for the then current Term. Perpetual licenses shall be pro-rated over a 36-month term. Altair shall have no obligation under this Section 7, however, if the alleged infringement arises from Altair's compliance with specifications or instructions prescribed by Licensee, modification of the Software by Licensee, use of the Software in combination with other software not provided by Altair and which use is not specifically described in the Documentation, and if Licensee is not using the most current version of the Software, if such alleged infringement would not have occurred except for such exclusions listed here. This section 7 states Altair's entire liability to Licensee in the event a Claim is made. No indemnification is made for Supplier and/or ISV Software.

8. LIMITATION OF REMEDIES AND LIABILITY. Licensee's exclusive remedy (and Altair's sole liability) for Software that does not meet the warranty set forth in Section 6 shall be, at Altair's option, either (i) to correct the nonconforming Software within a reasonable time so that it conforms to the warranty; or (ii) to terminate this Agreement and refund to Licensee the license fees that Licensee has paid for the then current Term for the nonconforming Software; provided, however that Licensee notifies Altair of the problem in writing within the applicable Warranty Period when the problem first occurs. Any corrected Software shall be warranted in accordance with Section 6 for ninety (90) days after delivery to Licensee. The warranties hereunder are void if the Software has been improperly installed, misused, or if Licensee has violated the terms of this Agreement.

Altair's entire liability for all claims arising under or related in any way to this Agreement (regardless of legal theory), shall be limited to direct damages, and shall not exceed, in the aggregate for all claims, the license and maintenance fees paid under this Agreement by Licensee in the 12 months prior to the claim on a prorated basis, except for claims under Section 7. ALTAIR AND ITS SUPPLIERS AND ISVS SHALL NOT BE LIABLE TO LICENSEE OR ANYONE ELSE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING HEREUNDER (INCLUDING LOSS OF PROFITS OR DATA, DEFECTS IN DESIGN OR PRODUCTS CREATED USING THE SOFTWARE, OR ANY INJURY OR DAMAGE RESULTING FROM SUCH DEFECTS, SUFFERED BY LICENSEE OR ANY THIRD PARTY) EVEN IF ALTAIR OR ITS SUPPLIERS OR ITS ISVS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Licensee acknowledges that it is solely responsible for the adequacy and accuracy of the input of data, including the output generated from such data, and agrees to defend, indemnify, and hold harmless Altair and its Suppliers and ISVs from any and all claims, including reasonable attorney's fees, resulting from, or in connection with Licensee's use of the Software. No
Appendix A: License Agreement

action, regardless of form, arising out of the transactions under this Agreement may be brought by either party against the other more than two (2) years after the cause of action has accrued, except for actions related to unpaid fees.

9. UNITED STATES GOVERNMENT RESTRICTED RIGHTS. This section applies to all acquisitions of the Products by or for the United States government. By accepting delivery of the Products except as provided below, the government or the party procuring the Products under government funding, hereby agrees that the Products qualify as “commercial” computer software as that term is used in the acquisition regulations applicable to this procurement and that the government's use and disclosure of the Products is controlled by the terms and conditions of this Agreement to the maximum extent possible. This Agreement supersedes any contrary terms or conditions in any statement of work, contract, or other document that are not required by statute or regulation. If any provision of this Agreement is unacceptable to the government, Vendor may be contacted at Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031; telephone (248) 614-2400. If any provision of this Agreement violates applicable federal law or does not meet the government's actual, minimum needs, the government agrees to return the Products for a full refund.

For procurements governed by DFARS Part 227.72 (OCT 1998), the Software, except as described below, is provided with only those rights specified in this Agreement in accordance with the Rights in Commercial Computer Software or Commercial Computer Software Documentation policy at DFARS 227.7202-3(a) (OCT 1998). For procurements other than for the Department of Defense, use, reproduction, or disclosure of the Software is subject to the restrictions set forth in this Agreement and in the Commercial Computer Software - Restricted Rights FAR clause 52.227-19 (June 1987) and any restrictions in successor regulations thereto.

Portions of Altair's PBS Professional Software and Documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c)(1)(ii) of the rights in the Technical Data and Computer Software clause in DFARS 252.227-7013, or in subdivision (c)(1) and (2) of the Commercial Computer Software-Restricted Rights clause at 48 CFR52.227-19, as applicable.

10. CHOICE OF LAW AND VENUE. This Agreement shall be governed by and construed under the laws of the state of Michigan, without regard to that state's conflict of laws principles except if the state of Michigan adopts the Uniform Computer Information Transactions Act drafted by the National Conference of Commissioners of Uniform State Laws as revised or amended as of June 30, 2002 (“UCITA”) which is specifically excluded. This Agreement shall not be governed by the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded. Each Party waives its right to a jury trial in the event of any dispute arising under or relating to this Agreement. Each party agrees that money damages may not be an adequate remedy for breach of the provisions of
this Agreement, and in the event of such breach, the aggrieved party shall be entitled to seek specific performance and/or injunctive relief (without posting a bond or other security) in order to enforce or prevent any violation of this Agreement.

11. [RESERVED]

12. Notice. All notices given by one party to the other under the Agreement or these Additional Terms shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All notices shall be deemed given upon actual receipt.

Written notice shall be made to:

**Altair:**

Licensee Name & Address:

Altair Engineering, Inc.  
1820 E. Big Beaver Rd  
Troy, MI 48083  
Attn: Tom M. Perring

13. TERM. For annual licenses, or Support provided for perpetual licenses, renewal shall be automatic for each successive year (“Renewal Term”), upon mutual written execution of a new Order Form. All charges and fees for each Renewal Term shall be set forth in the Order Form executed for each Renewal Term. All Software licenses procured by Licensee may be made coterminous at the written request of Licensee and the consent of Altair.

14. TERMINATION. Either party may terminate this Agreement upon thirty (30) days prior written notice upon the occurrence of a default or material breach by the other party of its obligations under this Agreement (except for a breach by Altair of the warranty set forth in Section 8 for which a remedy is provided under Section 10; or a breach by Licensee of Section 5 or Section 6 for which no cure period is provided and Altair may terminate this Agreement immediately) if such default or breach continues for more than thirty (30) days after receipt of such notice. Upon termination of this Agreement, Licensee must cease using the Software and, at Altair’s option, return all copies to Altair, or certify it has destroyed all such copies of the Software and Documentation.

15. GENERAL PROVISIONS. Export Controls. Licensee acknowledges that the Products may be subject to the export control laws and regulations of the United States and other countries, and any amendments thereof. Licensee agrees that Licensee will not directly or indirectly export the Products into any country or use the Products in any manner except in compliance with all applicable U.S. and other countries export laws and regulations. Notice. All notices given by one party to the other under this Agreement shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All
Appendix A: License Agreement

notices shall be deemed given upon actual receipt. **Assignment.** Neither party shall assign this Agreement without the prior written consent of other party, which shall not be unreasonably withheld. All terms and conditions of this Agreement shall be binding upon and inure to the benefit of the parties hereto and their respective successors and permitted assigns. **Waiver.** The failure of a party to enforce at any time any of the provisions of this Agreement shall not be construed to be a waiver of the right of the party thereafter to enforce any such provisions. **Severability.** If any provision of this Agreement is found void and unenforceable, such provision shall be interpreted so as to best accomplish the intent of the parties within the limits of applicable law, and all remaining provisions shall continue to be valid and enforceable. **Headings.** The section headings contained in this Agreement are for convenience only and shall not be of any effect in constructing the meanings of the Sections. **Modification.** No change or modification of this Agreement will be valid unless it is in writing and is signed by a duly authorized representative of each party. **Conflict.** In the event of any conflict between the terms of this Agreement and any terms and conditions on a Licensee Purchase Order or comparable document, the terms of this Agreement shall prevail. Moreover, each party agrees any additional terms on any Purchase Order or comparable document other than the transaction items of (a) item(s) ordered; (b) pricing; (c) quantity; (d) delivery instructions and (e) invoicing directions, are not binding on the parties. In the event of a conflict between the terms of this Agreement, and the Additional Terms, the Agreement shall take precedence. **Entire Agreement.** This Agreement, the Additional Terms, and the Order Form(s) attached hereto constitute the entire understanding between the parties related to the subject matter hereto, and supersedes all proposals or prior agreements, whether written or oral, and all other communications between the parties with respect to such subject matter. This Agreement may be executed in one or more counterparts, all of which together shall constitute one and the same instrument. **Execution.** Copies of this Agreement executed via original signatures, facsimile or email shall be deemed binding on the parties.
Appendix B: Third-party Legal Notices

The following copyrights apply:

Legal Notice for avltree

avltree - AVL index routines by Gregory Tseytin.
Copyright (c) 2000 Gregory Tseytin <tseyting@acm.org>
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permit-
ted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of condi-
tions and the following disclaimer as the first lines of this file unmodified.

2. Redistributions in binary form must reproduce the above copyright notice, this list of
conditions and the following disclaimer in the documentation and/or other materials pro-
vided with the distribution.

THIS SOFTWARE IS PROVIDED BY Gregory Tseytin "AS IS" AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WAR-
RANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.
Appendix B: Third-party Legal Notices

IN NO EVENT SHALL Gregory Tseytin BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Legal Notice for freebsd

Copyright (c) 1988, 1993 The Regents of the University of California. All rights reserved.

This code is derived from software written by Ken Arnold and published in UNIX Review, Vol. 6, No. 8.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement:
   This product includes software developed by the University of California, Berkeley and its contributors.

4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
Appendix B: Third-party Legal Notices

AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


Legal Notice for Cray Software

Copyright (c) 2003-2010 Cray Inc.
All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name Cray Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Appendix B: Third-party Legal Notices

Legal Notice for libical

The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Original Code is libical.

The Initial Developer of the Original Code is Eric Busboom.

All Rights Reserved.

Contributor(s): See individual source files.

Legal Notice for Expat

Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd and Clark Cooper

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Appendix B: Third-party Legal Notices

Legal Notice for hwloc

Copyright 2009 CNRS
 Copyright 2009 INRIA. All rights reserved.
 Copyright 2009 University of Bordeaux
 Copyright 2009 Cisco Systems, Inc. All rights reserved.
 See COPYING in top-level directory.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR `AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Legal Notice for editline

Libedit license (BSD style): http://sourceforge.net/projects/libedit/

Copyright (c) 1992, 1993 The Regents of the University of California. All rights reserved.
This code is derived from software contributed to Berkeley by Christos Zoulas of Cornell University.
Appendix B: Third-party Legal Notices

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

MinGwEditLine: http://sourceforge.net/projects/mingweditline/

A BSD-licensed EditLine API implementation for the native Windows Console

Copyright © 2010-2012 Paolo Tosco

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of MinGWEditLine nor the name of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
Appendix B: Third-party Legal Notices

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Appendix B: Third-party Legal Notices
Index

S$logevent MoM parameter 870
$restrict_user 668
$restrict_user_exceptions 668
$restrict_user_maxsysid 668
.rhosts 667

A
accelerator 780
accelerator_memory 780
accelerator_model 780
accept an action 417
access
  by group 648
  by user 648
  effect of flatuid 667
control lists 648
from host 648
to a queue 647
to a reservation 647
to the Server 647
Accounting 866
  log 864
acl_group_enable
  queue attribute 659
acl_groups
  queue attribute 659
acl_host_enable

ACLs 648
  default behavior 649
  format 649
  group 650
  host 650
  matching entry 652
  modifying behavior 649
  overrides 666
  removing entity 654
  rules for creating 654
  user 650
  who can create 656
action 417
Advance and Standing Reservations 249
  advance reservation 249, 289
aggressive_provision 600
alarm 446
alt_id 844
AOE 597
aoe resource
defining 610
application license
floating 350
definition 290
floating externally-managed 350
application licenses 346
floating license PBS-managed 351
license units and features 347
overview 322
per-host node-locked example 354
types 346
Associating Vnodes with Multiple Queues 122
Authorized_Groups reservation attribute 660
Authorized_Hosts reservation attribute 660
Authorized_Users reservation attribute 660
Average CPU Usage Enforcement 397
average_cpufactor 397
average_percent_over 397
average_trialperiod 397
avoid_provision 600
B
backfill_prime 246
basic fairshare 171
batch requests 870
borrowing vnode 290, 340
built-in resource 290
C
checkpoint 897
preemption via 236
chunk 290
comment 61
Configuration
Server 17

932

Configuring MoM on an Altix 813
consumable resource 290
CPU 290
cpuaverage 397
cpuset_create_flags 814
cpuset_destroy_delay 814
cpusets
troubleshooting 816
cput 174
Creating Queues 21
creation of provisioning hooks 613
credential 676
casswitch 799, 800
current_aoe 611
custom resource 290
custom resources
application licenses 346
floating managed by PBS 351
overview 322
per-host node-locked 354
types 346
how to use 319
restart steps 332
scratch space
overview 323
Static Host-level 339
Static Server-level 336
cycle harvesting
ideal_load 150, 150
max_load 150, 150
D
decay 175
dedicated time 153
defining aoe resource 610
defining provisioning policy 615
Defining Resources for the Altix 342
degraded reservation 250
department 171
Deprecations 9
DIS 860
DNS 900
Index

do_not_span_psets 783
Dynamic Fit 212
Dynamic Resource Scripts/Programs 324

E
egroup 171
euser 171
Eligible Wait Time 155
eligible_time 155, 158
enabled 447
euser 171
event 447
event classes 869
events
  exechost_periodic 511
  execjob_begin 504
  execjob_end 507
  execjob_epilogue 508
  execjob_preterm 510
  execjob_prologue 505
  modifyjob 500
  movejob 501
  queuejob 500
  resvsub 512
  runjob 502
  exechost_periodic 499
  exechost_periodic events 511
  execjob_begin 498
  execjob_begin events 504
  execjob_end 499
  execjob_end events 507, 508
  execjob_epilogue 499
  execjob_prologue 499
  execjob_preterm 510
  execjob_prologue events 505
  execution event hooks 417
  exiting 155
  express_queue 230

F
failover and hooks 439
fair_share 171
Fairshare 170
fairshare 230
fairshare entities 171
fairshare ID 173
Fairshare Tree 170
fairshare_usage_res 174
Files
  pbs.conf 900
  Finished jobs 853
  flatuid Server attribute 667
  floating license
    definition 290
    example 350
    example of externally-managed 350

G
gethostname 676
global resource 291
Globus 17, 861, 861
  PBS_MANAGER_GLOBUS_SERVICE_E_PORT 861
  PBS_MOM_GLOBUS_SERVICE_PORT 861

  group
    access 648
    ACLs 650
    limit 291, 370
      generic 369
      individual 369

H
half_life 175
help, getting 911
History jobs 853
hooks
  alarm 446
  and failover 439
  creating empty hooks 441
Index

creation of provisioning 613
deleting 441
enabled 447
enabling and disabling 449
event 447
event types 430
exechost_periodic 511
exejob_begin 504
exejob_end 507, 508
exejob_preterm 510
exejob_prologue 505
modifyjob 500
movejob 501
queuejob 500
resvsub 512
runjob 502
exechost_periodic events 511
exejob_begin events 504
exejob_end events 507, 508
exejob_preterm events 510
exejob_prologue events 505
execution event 417
exporting 444
importing 443
interface objects 473
job attributes 458
log event class objects 487
modifyjob events 500
movejob events 501
non-job event 418
order 448
overview of creating 440
pbs.event() 492
pbs.exec_vnode 524
pbs.job 519
pbs.queue() 518
pbs.resv 527
pbs.server() 515
pbs.vchunk 526
pbs.vnode 528
pre-execution event 418
provisioning 598
queuejob events 500
reservation attributes 465
resources 466
resvsub events 512
runjob events 502
setting order of execution 449
setting timeout 450
setting trigger events 441, 448
simple how-to 420
type 448
user 448
vnode attributes 463
host
access 648
ACLs 650
host.equiv 667
host-based Authentication 677
HPCBP 12
I
ideal_load
    cycle harvesting 150, 150
indirect resource 291, 340
ineligible_time 155
InfiniBand 767
initial_time 155
instance 249
instance of a standing reservation 249
interface objects for hooks 473
J
job
    attributes in hooks 458
Job attributes
    alt_id 844
Job History 852
    Changing Settings 854
    Configuring 854
    Enabling 854
    Setting Duration 854
job that can never run 898
job_history_duration 854
job_history_enable 854

L
license
    application
        floating 350
        definition 290
limit 292, 369
    attributes 376
    cput 394
    file size 394
    generic group limit 291, 369
    generic project limit 369
    generic user limit 291, 369
    group limit 291, 370
    individual group limit 291, 369
    individual project limit 370
    individual user limit 291, 369
    overall 292, 369
    pcput 394
    pmem 394
    project limit 370
    pvmem 394
    user limit 293, 370
    walltime 394
limits
    generic and individual 372
    group 366
    overall limits 373
    project 366
    resource usage 366
    scope 371
    setting limits 378
    user 366
load_balancing 148
LoadLeveler 771
local resource 292
Location of MoM’s configuration files 47
log event classes 869
log events
    MoM 870
    Scheduler 871
    Server 870
log_events Server attribute 870
log_filter scheduler parameter 871
logging
    hooks log event class objects 487
M
Manager
    privilege 646
managers Server attribute 646
managing vnode 292, 340
master provisioning script 597, 611
master script 597, 611
matching ACL entry 652
max_concurrent_provision 615
max_group_res 389, 390
max_group_run 389, 390, 390, 391
max_group_run_soft 389, 390
max_load
    cycle harvesting 150, 150
max_queuable 390
max_queued 376
max_queued_res 377
max_run 376
max_run_res 376
max_run_res_soft 376
max_run_soft 376
max_running 389, 391
max_user_res 390, 390
max_user_res_soft 389, 390, 390, 390
max_user_run 390, 390, 390, 390
max_user_run_soft 390, 390
max_walltime 270
memory-only vnode 292
min_walltime 270
modifyjob 497
modifyjob events 500
MoM
    log events 870
Index

restarting 909
movejob 498
movejob events 501
MPI, LAPI 771
MPI_USE_IB 767
mpiexec 766
multihost placement sets 214

N
naccelerators 781
natural vnode 38, 38
ncchunk 781
ncpus*walltime 174
netwins
  calculating 773
  Network Windows 772
  default 774
non-consumable resource 292
non-job event hooks 418
nonprimetime_prefix 246
normal_jobs 230
nrt_adapter_resources 777
nrt_load_table 777
nrt_status_adapter 777
nrt_unload_window 777

O
Operator
  privilege 645
operators Server attribute 646
order 448
overall limit 292, 369

P
password
  invalid 682, 897
  single-signon 682
  Windows 681, 682
pbs module 418
pbs.acl() 529
pbs.args() 534

pbs.checkpoint() 529
pbs.conf 895, 900
pbs.depend() 529
pbs.duration() 530
pbs.email_list() 530
pbs.event() 492
pbs.event().accept() 495
pbs.event().hook_alarm 493
pbs.event().hook_name 493
pbs.event().hook_type 493
pbs.event().reject() 495
pbs.event().requestor 493
pbs.event().requestor_host 493
pbs.event().type 486, 493
pbs.exec_host() 530
pbs.exec_vnode 524
pbs.exec_vnode() 530
pbs.get_local_nodename() 534
pbs.group_list() 531
pbs.hold_types() 531
pbs.job 519
pbs.job.delete() 523
pbs.job.id 520
pbs.job.in_ms_mom() 523
pbs.job.is_checkpointed() 523
pbs.job.rerun() 524
pbs.job.sort_formula() 531
pbs.join_path() 531
pbs.keep_files() 531
pbs.license_count() 531
pbs.logmsg() 534
pbs.mail_points() 531
pbs.node_group_key() 532
pbs.path_list() 532
pbs.place() 532
pbs.queue 518
pbs.queue() 518
pbs.queue. 518
pbs.queue.job() 519
pbs.queue.jobs() 519
pbs.queue.name 518
pbs.range() 532
Index

pbs.reboot() 495
pbs.resv 527
pbs.resv 527
pbs.resv.resvid 527
pbs.route_destinations() 532
pbs.select() 533
pbs.server() 515
pbs.server().job() 516
pbs.server().jobs() 516
pbs.server().name 515
pbs.server().queue() 516
pbs.server().resv() 517
pbs.server().resvs() 517
pbs.server().scheduler_restart_cycle() 517
pbs.server().vnode() 517
pbs.server().vnodes() 517
pbs.size() 533
pbs.software() 535
pbs.staging_list() 533
pbs.state_count() 534
pbs.user_list() 534
pbs.vchunk 526
pbs.version() 535
pbs.vnode 528
pbs_accounting_workload_mgmt 799
PBS_BATCH_SERVICE_PORT 860
PBS_BATCH_SERVICE_PORT_DIS 860
PBS_CONF_SYSLOG 861, 876
PBS_CONF_SYSLOGSEVR 861, 876
PBS_CPUSET_DEDICATED 767
PBS_ENVIRONMENT 860
PBS_EXEC 699, 860
PBS_GET_IBWINS 773
PBS_HOME 699, 860
PBS_HOME/sched_priv/resource_group 170
PBS_HOME/sched_priv/sched_config 171
PBS_HOME/sched_priv/usage 179
PBS_ibm_ib_default_networks 775
PBS_ibm_ib_max_instances 775
pbs_ iff 676, 895, 900
PBS_LOCALLOG 860, 875
PBS_MANAGER_GLOBUS_SERVICE_PORT 861
PBS_MANAGER_SERVICE_PORT 861
pbs_mkdirs 907
PBS_MOM_GLOBUS_SERVICE_PORT 861
PBS_MOM_HOME 699, 861
PBS_MOM_SERVICE_PORT 861
PBS_MPI_DEBUG 767
PBS_MPI_SGIARRAY 767
PBS_PRIMARY 700, 861
pbs_probe 907
PBS_RCP 861
pbs_rsub 660
PBS_SCHEDULER_SERVICE_PORT 861
PBS_SCP 861
PBS_SECONDARY 700, 861
PBS_SERVER 700, 861
PBS_START_MOM 700, 861
PBS_START_SCHED 700, 861
PBS_START_SERVER 700, 861
PBS_TMPDIR 862
PBScrayhost 299, 782
PBScraylabel 782
PBScraylabel 299
PBScray nid 299, 783
PBScrayorder 299, 783
PBScrayseg 299
pbsfs 179
pbsnodes 904
Placement Pool 212
Placement Set 211, 211
policy
    defining provisioning 615
preempt_order 229
preempt_prio 229
preempt_queue_prio 229
preempt_sort 229
preemption 228
preemption via checkpoint 236
Index

preemptive scheduling 228
preemptive_sched 229
pre-execution event hooks 418
Primary Server 861
prime_spill 246, 247
primetime_prefix 246
privilege
    manager 646
    operator 645
    user 645
project 370
project limit 370
generic 369
individual 370
project limits 366
provision_policy 600
provisioning
    creation of hooks 613
    defining policy 615
    hooks 598
    master script 611
    writing 611
overview 599
policy 600
rebooting 599
reservations 603
vnode selection 600
vnode states 604

Q
qdel 896
qhold 897
qmgr 17, 61, 895
qrun 896
qrls 682, 897
qstat 895, 900, 904, 904, 904
qsub 896
query_other_jobs Server attribute 645
queue 32
    access to a 647
    ACL 649
    attribute

acl_group_enable 659
acl_groups 659
acl_host_enable 659, 659
acl_hosts 659, 659
acl_user_enable 659, 659
acl_users 659, 659
reservation 662
queue_softlimits 230
queued jobs 370
queuejob 497
queuejob events 500
Quick Start Guide ix

R
rcp 677, 861
rebooting 599
reject an action 418
reservation
    access to a 647
    ACL 649
    advance 249, 289
    attribute
        Authorized_Groups 660
        Authorized_Hosts 660
        Authorized_Users 660
    attributes in hooks 465
    control of creation 648
    degraded 250
    instance 249
    queue 662
    reservation ID 250
    standing 249
    instance 249
    soonest occurrence 250
    reservations
        provisioning 603
    Resource 292
    resource
        built-in 290
        consumable 290
# Index

<table>
<thead>
<tr>
<th>Custom</th>
<th>290</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indirect</td>
<td>291, 340</td>
</tr>
<tr>
<td>Non-consumable</td>
<td>292</td>
</tr>
<tr>
<td>Shared</td>
<td>292, 340</td>
</tr>
<tr>
<td>Resource limits</td>
<td>366</td>
</tr>
<tr>
<td>Resource usage limits</td>
<td>366</td>
</tr>
<tr>
<td>Resources in Hooks</td>
<td>466</td>
</tr>
<tr>
<td>Restart</td>
<td>909</td>
</tr>
<tr>
<td>Custom Resources</td>
<td>332</td>
</tr>
<tr>
<td>Restarting MoM</td>
<td>909</td>
</tr>
<tr>
<td>Scheduler</td>
<td>910</td>
</tr>
<tr>
<td>Server</td>
<td>909</td>
</tr>
<tr>
<td>Restrict User</td>
<td>669</td>
</tr>
<tr>
<td>Restrict User Exceptions</td>
<td>669</td>
</tr>
<tr>
<td>Restrict User Maxsysid</td>
<td>669</td>
</tr>
<tr>
<td>Resv Enable</td>
<td>660</td>
</tr>
<tr>
<td>Resv Enable Server Attribute</td>
<td>648</td>
</tr>
<tr>
<td>Resvsub</td>
<td>499</td>
</tr>
<tr>
<td>Resvsub Events</td>
<td>512</td>
</tr>
<tr>
<td>Roles</td>
<td>644</td>
</tr>
<tr>
<td>Rsh</td>
<td>677</td>
</tr>
<tr>
<td>Run Time</td>
<td>155</td>
</tr>
<tr>
<td>Runjob</td>
<td>498</td>
</tr>
<tr>
<td>Runjob Events</td>
<td>502</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales, PBS</td>
</tr>
<tr>
<td>Sales, Support</td>
</tr>
<tr>
<td>Scheduler</td>
</tr>
<tr>
<td>Logging Events</td>
</tr>
<tr>
<td>Restarting</td>
</tr>
<tr>
<td>Scp</td>
</tr>
<tr>
<td>Scratch Space</td>
</tr>
<tr>
<td>Dynamic</td>
</tr>
<tr>
<td>Host-Level</td>
</tr>
<tr>
<td>Server-Level</td>
</tr>
<tr>
<td>Static</td>
</tr>
<tr>
<td>Host-Level</td>
</tr>
<tr>
<td>Server-Level</td>
</tr>
</tbody>
</table>

| Master Provisioning | 611 |
| Writing Provisioning | 611 |
| Secondary Server | 861 |
| Server Access to the | 647 |
| ACL | 649 |
| Attribute | |
| Acl Host Enable | 659, 659 |
| Acl Hosts | 659, 659 |
| Acl User Enable | 659, 659 |
| Acl Users | 659, 659 |
| Flatuid | 667 |
| Log Events | 870 |
| Managers | 646 |
| Operators | 646 |
| Query Other Jobs | 645 |
| Resv Enable | 648 |
| Log Events | 870 |
| Parameters | 17 |
| Recording Configuration | 17 |
| Restarting | 909 |
| Server Softlimits | 230 |
| Setting Hook Trigger Events | 441, 448 |
| Setting Limits | 378 |
| Shared Resource | 292, 340 |
| Shares | 170 |
| Sighup | 864 |
| Single Signon Password Enable | 681, 682 |
| Single-Signon | 682 |
| Soonest Occurrence | 250 |
| Sort Key | 175 |
| Ssh | 677 |
| Standing Reservation | 249 |
| Starving Jobs | 230 |
| States | |
| Vnodes and Provisioning | 604 |
| Static Fit | 212 |
| Strict Ordering and Backfilling | 282 |
| Sun Solaris-Specific Memory Enforcement | 396 |
| Support Team | 904, 911, 911 |
| Syslog | 875 |
Index

T
Task placement 211
Tips & Advice 911
tree percentage 174
troubleshooting
cpusets 816
type 448
Type codes 870

U
unknown node 170
unknown_shares 171
Unset Resources 199
usage limits 366
user 448
access 648
ACLs 650
privilege 645
roles 644
User Guide ix, 897, 897
user limit 293, 370
generic 369
individual 369
user limits 366

V
Version 894
Virtual Nodes 37
vnode 37
attributes in hooks 463
borrowing 290, 340
managing 292, 340
memory-only 292
natural 38
selection for provisioning 600
states and provisioning 604
vnode_def_additive 784
vnodes
hosts and nodes 38

W
Windows
errors 899
password 681, 682, 897
writing provisioning script 611

X
X forwarding 863
xauth 863