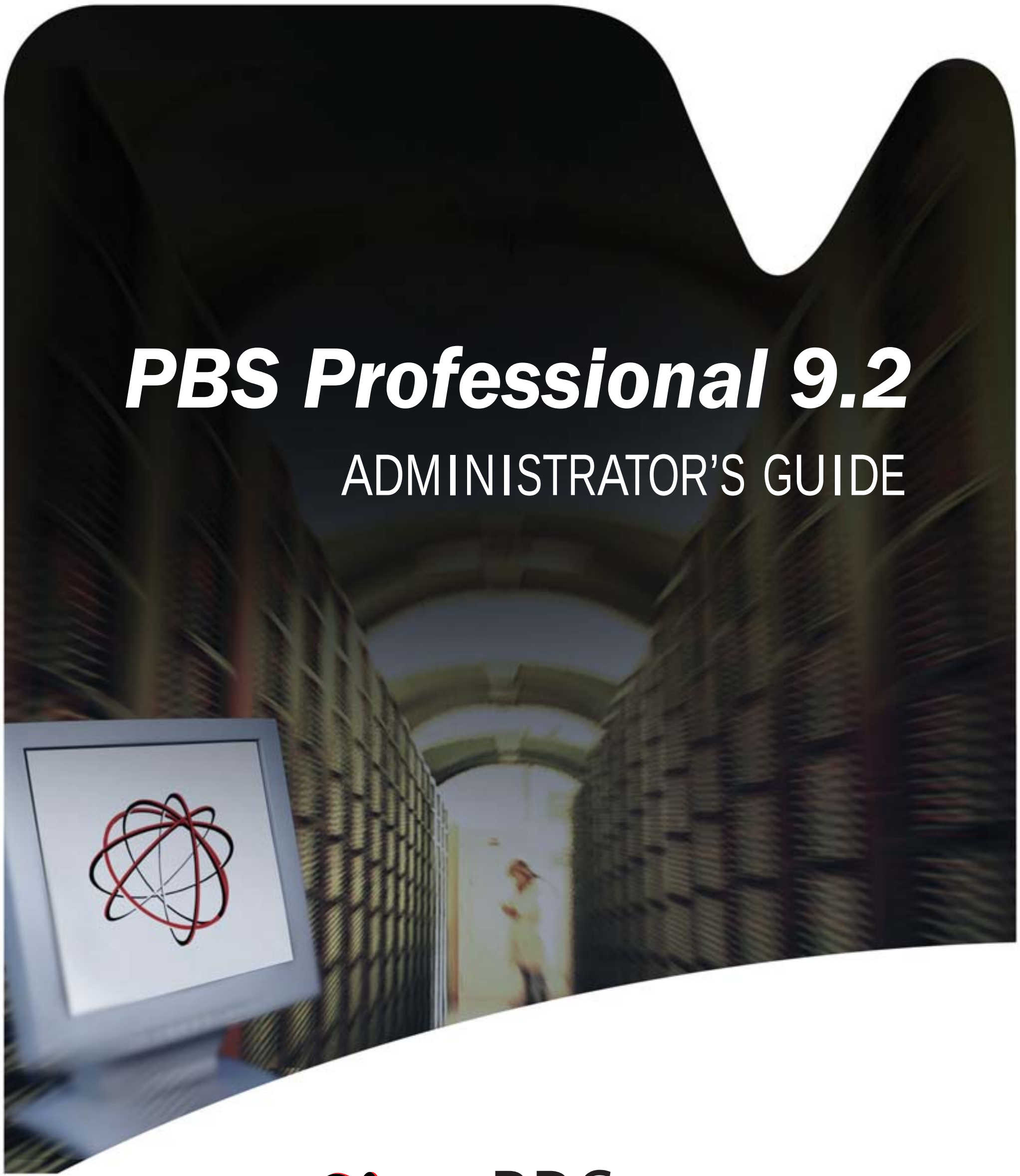
PBS Professional 9.2

ADMINISTRATOR'S GUIDE



PBS GridWorks™

Enabling On-Demand Computing™

A division of Altair

Altair®

PBS Professional™

9.2

Administrator's Guide

UNIX®, Linux® and Windows®

PBS Professional™ Administrator's Guide

Altair® PBS Professional™ 9.2, Updated: 5/15/08

Edited by: Anne Urban

For more information, copies of these books, and for product sales, contact Altair at:

Web: www.altair.com www.pbspro.com

Email: sales@pbspro.com

Technical Support

Table 1:

Location	Telephone	e-mail
North America	+1 248 614 2425	pbssupport@altair.com
China	+86 (0)21 5393 0011	support@altair.com.cn
France	+33 (0)1 4133 0990	francesupport@altair.com
Germany	+49 (0)7031 6208 22	hwsupport@altair.de
India	+91 80 658 8540 +91 80 658 8542	pbs-support@india.altair.com
Italy	+39 0832 315573 +39 800 905595	support@altairtorino.it
Japan	+81 3 5396 1341	pbs@altairjp.co.jp
Korea	+82 31 728 8600	support@altair.co.kr
Scandinavia	+46 (0)46 286 2050	support@altair.se
UK	+44 (0) 2476 323 600	support@uk.altair.com

Table of Contents

Acknowledgements

PBS Professional is the enhanced commercial version of the PBS software originally developed for NASA. The NASA version had a number of corporate and individual contributors over the years, for which the PBS developers and PBS community are most grateful. Below we provide formal legal acknowledgements to corporate and government entities, then special thanks to individuals.

The NASA version of PBS contained software developed by NASA Ames Research Center, Lawrence Livermore National Laboratory, and MRJ Technology Solutions. In addition, it included software developed by the NetBSD Foundation, Inc., and its contributors, as well as software developed by the University of California, Berkeley and its contributors.

Other contributors to the NASA version of PBS include Bruce Kelly and Clark Streeter of NERSC; Kent Crispin and Terry Heidelberg of LLNL; John Kochmar and Rob Pennington of *Pittsburgh Supercomputing Center*; and Dirk Grunwald of *University of Colorado, Boulder*. The ports of PBS to the Cray T3e was funded by *DoD USAERDC*, Major Shared Research Center; the port of PBS to the Cray SV1 was funded by DoD MSIC.

No list of acknowledgements for PBS would possibly be complete without special recognition of the first two beta test sites. Thomas Milliman of the *Space Sciences Center* of the *University of New Hampshire* was the first beta tester. Wendy Lin of *Purdue University* was the second beta tester and continues to provide excellent feedback on the product.

Preface

Intended Audience

This document provides the system administrator with the information required to install, configure, and manage PBS Professional (PBS). PBS is a workload management system that provides a unified batch queuing and job management interface to a set of computing resources.

Related Documents

The following publications contain information that may also be useful in the management and administration of PBS.

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

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

PBS Professional User's Guide:
Explains how to use the user commands and graphical user interface to submit, monitor, track, delete, and manipulate jobs.

PBS Professional External Reference Specification: Discusses in detail the PBS application programming interface (API), security within PBS, and intra-component communication.

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`	If a PBS command can be abbreviated (such as subcommands to `qmgr`) the shortest acceptable abbreviation is underlined.
`command`	This fixed width font is used to denote literal commands, file-names, error messages, and program output.
`input`	Literal user input is shown in this bold, fixed-width font.
`manpage(x)`	Following UNIX tradition, manual page references include the corresponding section number in parentheses appended to the manual page name.
`terms`	Words or terms being defined, as well as variable names, are in italics.

Chapter 1

New Features

This chapter presents information needed prior to installing PBS. First, a reference to new features in this release of PBS Professional is provided. Next is the information necessary to make certain planning decisions.

1.1 New Features in PBS Professional 9.2

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 following is a list of major new features.

Administrator's Guide	Permissions for custom resources. See section 2.10.9 “Resource Flags for Resource Permissions” on page 71.

Administrator's Guide	Extension to tunable formula. See section 4.7.2 "Tunable Formula for Computing Job Priorities" on page 194.
Administrator's Guide	Per-job staging and execution directories. See section 6.14 "The Job's Staging and Execution Directories" on page 330.
Administrator's Guide	Support for standing reservations. See section 4.8 "Advance and Standing Reservations" on page 204.
Administrator's Guide	Eligible wait time for jobs. See section 4.1.2 "Eligible Wait Time for Jobs" on page 158.
Installation & Upgrade Guide	Support for the Cray XT. See section 4.9 "Installing PBS on the Cray XT" on page 59 in the PBS Professional Installation & Upgrade Guide and section 6.9.2 "Installing PBS To Upgrade on the Cray XT" on page 149 in the PBS Professional Installation & Upgrade Guide.
Installation & Upgrade Guide	Support for Altix ICE/XE. See section 4.8.3 "Installing PBS on the SGI ICE/XE" on page 53 in the PBS Professional Installation & Upgrade Guide.

1.1.1 Resource Permissions for Custom Resources

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 `qstat`. See section 2.10.9 "Resource Flags for Resource Permissions" on page 71.

1.1.2 Extension to Tunable Formula

The tunable formula has been extended to include parentheses, exponentiation, division, and unary plus and minus. See section 4.7.2 "Tunable Formula for Computing Job Priorities" on page 194.

1.1.3 Eligible Wait Time for Jobs

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 tunable formula. Jobs have two new attributes, eligible_time and accrue_type, which indicates what kind of wait time the job is accruing. See section 4.7.3 “Eligible Wait Time for Jobs” on page 200.

1.1.4 Job Staging and Execution Directories

PBS now provides per-job staging and execution directories. Jobs have new attributes `sandbox` and `jobdir`, the MOM has a new option `$jobdir_root`, and there is a new environment variable called `PBS_JOBDIR`. If the job’s `sandbox` attribute is set to `PRIVATE`, PBS creates a job-specific staging and execution directory. If the job’s `sandbox` attribute is unset or is set to `HOME`, PBS uses the user’s home directory for staging and execution, which is how previous versions of PBS behaved. If MOM’s `$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 6.14 “The Job’s Staging and Execution Directories” on page 330.

1.1.5 Standing Reservations

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 “Advance and Standing Reservations” on page 204.

1.2 Changes in Previous Releases

1.2.1 New Server Attribute for Tunable Formula

The new server attribute “job_sort_formula” is used for sorting jobs according to a site-defined formula. See section 9.7.2 “Tunable Formula for Computing Job Priorities” on page 342.

1.2.2 Change to sched_config

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

1.2.3 Change to Licensing

PBS now depends on a FLEX/Altair license server that will hand out licenses to be assigned to PBS jobs. See section 5.1 “FLEX Licensing Feature” on page 87 in the PBS Professional Installation & Upgrade Guide. A site can still use a trial license. See section 5.2 “Trial Licenses” on page 88 in the PBS Professional Installation & Upgrade Guide. PBS Professional versions 8.0 and below will continue to be licensed using the proprietary licensing scheme.

1.2.4 Installing With FLEX Licensing

You must install and configure the FLEXlm license server before installing and configuring PBS. See section 4.1 “Overview of Installing PBS” on page 37 in the PBS Professional Installation & Upgrade Guide.

1.2.5 Unset Host-level Resources Have Zero Value

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 2.10.2 “Unset Resources” on page 62.

1.2.6 Better Management of Resources Allocated to Jobs

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.

1.2.7 Support for Large Page Mode on AIX

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

1.3 Deprecations

The **sort_priority** option to job_sort_key is deprecated and is replaced with the job_priority option.
The **-l nodes=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.
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.

Chapter 2

Configuring the Server

The next three chapters will walk you through the process of configuring the Server, the MOMs and the scheduling policy. Further configuration may not be required as the default configuration may completely meet your needs. However, you are advised to read this chapter to determine if the default configuration is indeed complete for you, or if any of the optional settings may apply.

2.1 New Server Features

2.1.1 Permissions for Custom Resources

You can set permissions for custom resources. See section 2.10.9 "Resource Flags for Resource Permissions" on page 71.

2.2 The qmgr Command

The PBS manager command, `qmgr`, provides a command-line interface to the PBS Server. The qmgr command can be used by anyone to list or print attributes. Operator privilege is required to be able to set or unset vnode, queue or server attributes. Manager privilege is required to create or delete queues or vnodes. The qmgr command will not display attributes which are unset, i.e. are at their default value.

Most of a vnode's attributes may be set using qmgr. However, some **must** be set on the individual execution host in local vnode definition files, NOT by using qmgr. Those that must be set on the execution host this way are

sharing
ncpus
mem
vmem

An example of the way to do this (in this case, changing the "sharing" attribute for a vnode named V10) uses the script "change_sharing". See section 3.2.1 "Creation of Site-defined MOM Configuration Files" on page 109.

```
# cat change_sharing
$configversion 2
V10:  sharing = ignore_excl
# . /etc/pbs.conf
# $PBS_EXEC/sbin/pbs_mom -s insert
ignore_excl change_sharing
# pkill -HUP pbs_mom
```

Do **not** set sharing, ncpus, mem, or vmem on a vnode via qmgr.

The `qmgr` command usage is:

qmgr [-a] [-c command] [-e] [-n] [-z] [server...]
qmgr --version

The available options, and description of each, follows.

Table 1:

Option	Action
-a	Abort `qmgr` on any syntax errors or any requests rejected by a Server.
-c command	Execute a single command and exit `qmgr`. The command must be enclosed in quote marks, e.g. `qmgr -c "print server"`
-e	Echo all commands to standard output.
-n	No commands are executed, syntax checking only is performed.
-z	No errors are written to standard error.
--version	The qmgr command returns its PBS version information and exits. This option can only be used alone.

If `qmgr` is invoked without the `-c` option and standard output is connected to a terminal, `qmgr` will write a prompt to standard output and read a directive from standard input.

Any attribute value set via **`qmgr`** containing commas, whitespace or the hashmark must be enclosed in double quotes. For example:

```
Qmgr: set node Vnode1 comment="Node will be
taken offline Friday at 1:00 for memory
upgrade."
Qmgr: active node vnode1,vnode2,vnode3
```

A command is terminated by a new line character or a semicolon (";") character. Multiple commands may be entered on a single line. A command may extend across lines by escaping the new line character with a backslash ("\"). Comments begin with the "#" character and continue to the end of the line. Comments and blank lines are ignored by `qmgr`. The syntax of each directive is checked and the appropriate request is sent to the Server(s). A `qmgr` directive takes one of the following forms (OP is the operation to be performed on the attribute and its value):

command server [names] [attr OP value[,...]]
command queue [names] [attr OP value[,...]]
command node [names] [attr OP value[,...]]
command sched [names] [attr OP value[,...]]

Where command is the sub-command to perform on an object. The commands are listed in the table below.

The object of the command can be explicitly named, as in"

```
qmgr -c "print queue <queue name>"
```

or can be specified before using the command, by making the object(s) active, for example:

```
qmgr -c "active Vnode1"
```

Only vnodes and queues can be created or deleted using qmgr.

You can specify the default server in a command by using "@default" instead of @<server name>. If you don't name a specific object, all objects of that type at the server will be affected.

For example, to print out all of the queue information for the default server:

```
qmgr -c "print queue @default"
```

Under Windows, use double quotes when specifying arguments to PBS commands, including `qmgr`.

Table 2:

Command	Explanation
active	Sets the objects that will be operated on in following commands. These objects remain active until the active command is used. Disregarded when an object is specified in a qmgr command.
create	Creates a new object; applies to queues and vnodes.

Table 2:

Command	Explanation
delete	Destroys an existing object; applies to queues and vnodes.
help	Prints command-specific help and usage information
list	Lists the current attributes and associated values of the object.
print	Prints settable queue and Server attributes in a format that will be usable as input to the `qmgr` command.
set	Defines or alters attribute values of the object.
unset	Clears the value of the attributes of the object. Note: this form does not accept an OP and value, only the attribute name.

Other `qmgr` syntax definitions follow:

Table 3:

Variable	`qmgr` Variable/Syntax Description
names	List of one or more names of specific objects. The name list is in the form: [name][@server][,name[@server]...] with no intervening white space. The name of an object is declared when the object is first created. If the name is `@server`, then all the objects of specified type at the Server will be affected.
attr	Specifies the name of an attribute of the object which is to be set or modified. The attributes of objects are described on the relevant attribute man page (e.g. `pbs_node_attributes(3B)`). If the attribute is one which consists of a set of resources, then the attribute is specified in the form: attribute_name.resource_name

Table 3:

Variable	`qmgr` Variable/Syntax Description
OP	An operation to be performed with the attribute and its value:
=	Set the value of the attribute. If the attribute has an existing value, the current value is replaced with the new value.
+=	Increase the value of the attribute by the amount specified. Used to append a string to a string array, for example “s s managers+=<manager name>”
-=	Decrease the value of the attribute by the amount specified. Used to remove a string from a string array, for example “s s managers-=<manager name>”
value	The value to assign to an attribute. If value includes white space, commas, square brackets or other special characters, such as “#”, the value string must be enclosed in quote marks (“ ”).

A few examples of the `qmgr` command follow. Commands can be abbreviated. The underlined letters are there to show which abbreviations can be used in place of complete words.

```
Qmgr: create node mars
Qmgr: set node mars
resources_available.ncpus=2
Qmgr: create node venus
Qmgr: set node mars resources_available.inner
= true
Qmgr: set node mars
resources_available.haslife= true
Qmgr: delete node mars
Qmgr: d n venus
```

2.2.1 qmgr Help System

The qmgr built-in help function, invoked using the "help" sub-command, is illustrated by the next example which shows that requesting usage information on qmgr's set command produces the following output.

```
qmgr
Qmgr: help set
Syntax: set object [name][,name...]
attribute[.resource] OP value
```

Objects can be "server" or "queue", "node"

The "set" command sets the value for an attribute on the specified object. If the object is "server" and name is not specified, the attribute will be set on all the servers specified on the command line. For multiple names, use a comma separated list with no intervening whitespace.
Examples:

```
set server s1 max_running = 5
set server managers = root
set server managers += susan
set node n1,n2 state=down
set queue q1@s3 resources_max.mem += 5mb
set queue @s3 default_queue = batch
```

Custom resources can be made invisible to users or unalterable by users via resource permission flags. See section 2.10.9 "Resource Flags for Resource Permissions" on page 71. A user will not be able to print or list custom resource which have been made either invisible or unalterable.

2.3 Default Configuration

Server management consists of configuring the Server attributes, defining vnodes, and establishing queues and their attributes. The default configuration from the binary installation sets the minimum Server settings, and some recommended settings for a typical PBS complex. (The default Server configuration is shown below.) The subsequent sections in this chapter list, explain, and provide the default settings for all the Server's attributes for the default binary installation.

```
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
```

2.3.1 PBS Levels of Privilege

The `qmgr` command is subject to the three levels of privilege in PBS:

Manager, Operator, and user. In general, a "Manager" can do everything offered by `qmgr` (such as creating/deleting new objects like queues and vnodes, modifying existing objects, and changing attributes that affect policy). The "Operator" level is more restrictive. Operators cannot create new objects nor modify any attribute that changes scheduling policy. See "operators" on page 28. A "user" can view, but cannot change, Server configuration information. For example, the `help, list` and `print` subcommands of `qmgr` can be executed by the general user. Creating or deleting a queue requires PBS Manager privilege. Setting or unsetting `Server` or `queue` attributes (discussed below) requires PBS Operator or Manager privilege. Specifically, Manager privilege is required to create and delete queues or vnodes, and set/alter/unset the following attributes:

Table 4: Attributes Requiring Manager Privilege to Set or Alter

Server	Queue	Vnode
acl_hosts	alt_route	comment
acl_host_enable	from_route_queue	Mom
acl_resv_groups	require_cred	no_multinode_jobs
acl_resv_group_enable	require_cred_enable	pnames
acl_resv_hosts	route_destinations	queue
acl_resv_host_enable		resv_enable
acl_resv_users		
acl_resv_user_enable		
acl_roots		
acl_users		
acl_user_enable		
default_node		
flatuid		
mail_from		
managers		
operators		

Table 4: Attributes Requiring Manager Privilege to Set or Alter

Server	Queue	Vnode
query_other_jobs		
require_cred		
require_cred_enable		
resv_enable		

For details on setting these levels of privilege, see the `managers` and `operators` Server attributes, discussed in "Server Configuration Attributes" on page 18; for security-related aspects of PBS privilege, see section 6.8.7 "External Security" on page 302.)

2.4 The Server's Nodes File

The server creates a file of the nodes managed by PBS. This nodes 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 nodes 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 nodes file created for each job. See section 6.10.1 "The PBS_NODEFILE" on page 307.

2.5 Hard and Soft Limits

Hard limits cannot be exceeded. Soft limits can be exceeded, but make the user's jobs eligible for preemption. Hard and soft limits can be set for the number of jobs a user can run, or usage of a particular resource. Hard and soft limits can also be set for a group, both for number of jobs running and amount of resources used. Soft limits are only used with preemption.

Example of setting user run limits:

```
s q <queue_name> max_user_run=5
s q <queue_name> max_user_run_soft=4
```

Once a user has exceeded their soft limit, their jobs are eligible for preemption. In this example, a soft limit means that when user A has reached a `max_user_run_soft of 4`, their 5th job will still run, but their 6th will not. However, all of user A's jobs are now eligible to be preempted by another user who is under their limits. If it is necessary in order to run the other user's jobs, one of user A's jobs will be preempted, then another, until user A is no longer over their soft limit.

Hard and soft resource limits work the same way. When a user exceeds a resource soft limit, that user's jobs are eligible for preemption.

Example of setting user resource limits:

```
s q <queue_name> max_user_res.mem=200gb
s q <queue_name> max_user_res_soft.mem=100gb
```

The user will not be allowed to start jobs which would exceed the hard resource limit. So if a user's first job only uses 100GB of memory, that job will run. If the user then submits a second job that requests 200GB of memory, that job will not start while the first one is running. If a job is submitted that would exceed that limit by itself, that job stays queued indefinitely.

Note that `max_user_run_soft` and `max_user_res_soft` can only be set at the server and queue levels.

For more information on soft limits, see the `pbs_server_attributes(7B)` and `pbs_queue_attributes(7B)` man pages. See also the discussion of scheduling parameters using soft limits in "Enabling Preemptive Scheduling" on page 214.

2.6 Server Configuration Attributes

This section explains all the available Server configuration attributes and gives the default values for each. These attributes are set via the `qmgr` command.

`acl_host_enable`

When `true` directs the Server to use the `acl_hosts` access control lists. Requires Manager privilege to set or alter.
Format: boolean
Default value: false = disabled
Qmgr: set server acl_host_enable=true

`acl_hosts`

List of hosts which may request services from this Server. This list contains the fully qualified network name of the hosts. Local requests, i.e. from the Server's host itself, are always accepted even if the host is not included in the list. Wildcards ("*") may be used for hostnames. See also `acl_host_enable`.
Format: "[+|-]hostname.domain[,...]"
Default value: all hosts
Qmgr: set server acl_hosts=*.domain.com
Qmgr: set server acl_hosts="+*.domain.com,-*"
Qmgr: set server acl_hosts+=<host-name.domain.com>

`acl_resv_host_enable`

When true directs the Server to use the `acl_resv_hosts` access control list. Requires Manager privilege to set or alter.
Format: boolean
Default value: false = disabled
Qmgr: set server acl_resv_host_enable=true

`acl_resv_hosts`

List of hosts which may request reservations from this server. This list contains the network name of the hosts. Local requests, i.e. from the Server's host itself, are always accepted even if the host is not

included in the list. Wildcards (“*”) may be used for hostnames. Requires Manager privilege to set or alter. See also `acl_resv_enable`.
Format: "[+|-]hostname.domain[,...]”
Default value: all hosts
To put all hosts in the domain on the list of those that can request reservations:
Qmgr: set server acl_resv_hosts=*.domain.com
To put a host on the list of hosts not allowed to request reservations:
Qmgr: set server acl_resv_hosts+=-host.domain.com
To add to list of allowed hosts:
Qmgr: set server acl_resv_hosts+=host.domain.com
To remove from list of allowed hosts:
Qmgr: set server acl_resv_hosts-=host.domain.com

acl_resv_group_enable

If true directs the Server to use the reservation group ACL `acl_resv_groups`. Requires Manager privilege to set or alter. Format: boolean
Default value: false = disabled
Qmgr: set server acl_resv_group_enable=true

`acl_resv_groups`

List which allows or denies accepting reservations owned by members of the listed groups. The groups in the list are groups on the Server host, not submitting hosts. See also `acl_resv_group_enable`.
Format: “[+|-]group_name[,...]”
Default value: all groups allowed
Qmgr: set server acl_resv_groups=”blue,green”

`acl_resv_user_enable`

If true, directs the Server to use the `acl_resv_users` access list. Requires Manager privilege to set or alter.
Format: boolean
Default value: disabled
Qmgr: set server acl_resv_user_enable=true

`acl_resv_users`

A single list of users allowed or denied the ability to make reservation requests of this Server. Requires Manager privilege to set or alter. See also `acl_resv_user_enable`. Manager privilege overrides user access restrictions. The order of the elements in the list is important. The list is searched, starting at the beginning, for a match. The first match encountered in the list is accepted and terminates processing. Therefore, to allow all users except for some, the list of denied users should be put at the front of the list, followed by the set of allowed users. When usernames are added to the list, they are appended to the end of the list.
Format: “[+|-]user[@host][,...]”
Default value: all users allowed
To set list of allowed users:
Qmgr: set server acl_resv_users=”-bob,-tom,joe,+”
To add to list of allowed users:
Qmgr: set server acl_resv_users+=nancy@terra
To remove from list of allowed users:
Qmgr: set server acl_resv_users-=joe
To remove from list of disallowed users:
Qmgr: set server acl_resv_users-=-joe
To add to list of disallowed users:
Qmgr: set server acl_resv_users+=-mary

`acl_user_enable`

When true directs the Server to use the Server level `acl_users` access list. Requires Manager privilege to set or alter.
Format: boolean
Default value: disabled
Qmgr: set server acl_user_enable=true

`acl_users`

A single list of users allowed or denied the ability to make any requests of this Server. Requires Manager privilege to set or alter. See also `acl_user_enable`. Manager privilege overrides user access restrictions. The order of the elements in the list is important. The list is searched, starting at

the beginning, for a match. The first match encountered in the list is accepted and terminates processing. Therefore, to allow all users except for some, the list of denied users should be put at the front of the list, followed by the set of allowed users. When usernames are added to the list, they are appended to the end of the list.
Format: "[+|-]user[@host][,...]"
Default value: all users allowed
To set list of allowed users:
Qmgr: set server acl_users="-bob,-tom,joe,+"
To add to list of allowed users:
Qmgr: set server acl_users+=nancy@terra
To remove from list of allowed users:
Qmgr: set server acl_users-=joe
To add to list of disallowed users:
Qmgr: set server acl_users+=-mary

`acl_roots` List of superusers who may submit to and execute jobs at this Server. If the job execution ID is zero (0), then the job owner, root@host, must be listed in this access control list or the job is rejected. See **`acl_users`** for syntax.
Format: "[+|-]user[@host][,...]"
Default value: no root jobs allowed
Qmgr: set server acl_roots=root@host

`comment` A text string which may be set by the Scheduler or other privileged client to provide information to PBS users.
Format: any string
Default value: none
Qmgr: set server comment="Planets Cluster"

`default_chunk`

Defines default elements of chunks for all jobs on this server. All jobs will inherit default chunk elements for elements not set at submission time. Jobs moved to this server from another server will lose their old defaults and inherit these.
Format: resource specification format,

e.g. “default_chunk.resource= \
value,default_chunk.resource=value, ...”
Qmgr: `set server default_chunk.mem= 100mb,default_chunk.ncpus=1`

It is strongly advised not to set "default_chunk.ncpus=1" to zero. The attribute may be set to a higher value if appropriate.

default_qdel_arguments

String containing argument to qdel. Argument is “-Wsuppress_mail=<N>”. Settable by the administrator. Overridden by arguments given on the command line. Default: none
Example of setting value:
Qmgr: set server default_qdel_arguments = "-Wsuppress_email = 3"

`default_qsub_arguments`

String containing any valid arguments to qsub. Settable by the administrator. Overridden by arguments given on the command line and in script directives. Job resources inherited 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`.

Default: none
Example of setting value:
Qmgr: set server
default_qsub_arguments = "-m n -r n"

`default_queue`

The queue which is the target queue when a request does not specify a queue name.
Format: a queue name.
Default value: workq
Qmgr: set server default_queue=workq

`eligible_time_enable`

Controls starving behavior. When set to true, the value of the job's eligible_time attribute is used for its starving time. When set to false, the job's starving time is calculated as now() - etime.
Viewable via qstat by job owner, Operator and Manager.
Settable only by manager, and read-only for job owner and operator. See section 4.7.3 "Eligible Wait Time for Jobs" on page 200.
Default: false.

Qmgr> set server eligible_time_enable=True

`flatuid`

Attribute which directs the Server to automatically grant authorization for a job to be run under the user name of the user who submitted the job even if the job was submitted from a different host. If not set `true`, then the Server will check the authorization of the job owner to run under that name if not submitted from the Server's host. See section 6.8.5 "User Authorization" on page 299 for usage and important caveats.
Format: boolean
Default value: false = disabled
Qmgr: set server flatuid=True

`job_sort_formula`

Formula for computing job priorities in the finest-granularity class given in section 4.7 "Job Priorities in PBS Professional" on page 193. If the attribute job_sort_formula is set, the scheduler will compute job priorities according to the formula. If it is unset, the scheduler computes this class of job priorities according to fairshare, if fairshare is enabled. If neither is defined, the scheduler uses job_sort_key. When the scheduler sorts jobs according to the formula, it computes a priority for each job, where that priority is the value produced by the formula. Jobs with a higher value get higher priority. To set the job_sort_formula attribute, use the qmgr command:

Qmgr> s s job_sort_formula = "<formula>"

The formula can be made up of any number of expressions, where expressions contain terms which are added, subtracted or multiplied. You cannot use division. Multiplication takes precedence over addition or subtraction. You cannot use two operators in a row. For example, "A+-B" is disallowed.

Terms can be:

Constants expressed as NUM or NUM.NUM:
[0-9]+'.'[0-9]+
The following attribute values:
queue_priority: value of priority attribute for queue in which job resides
job_priority: value of the job's priority attribute
fair_share_perc: percentage of fairshare tree for this job's entity
The following resources: (the amount requested, not used)
ncpus
mem
walltime
cput
Custom numeric job-wide resources: these must be alphanumeric with a leading alphabetic:
[a-zA-Z][a-zA-Z0-9_]*
This will represent the amount requested, not the amount used.
They must be of type long, float, or size.

Default: unset.
Can be set by Manager or Operator.

`log_events` A bit string which specifies the type of events which are logged; see also section 6.17 “Use and Maintenance of Logfiles” on page 353.
Format: integer

Default value: 511 (all events)
Qmgr: set server log_events=255

`mail_from` The email address used as the “from” address for Server-generated mail sent to users, as well as the address where email about important events and warnings will be sent. On Windows, must be a fully qualified mail address.
Format: string
Default value: adm
Qmgr: set server mail_from=boss@domain.com

`managers` List of users granted PBS Manager privileges. The hostname may be wildcarded by the use of an * character. Requires Manager privilege to set or alter.
Format:
“user@host.sub.domain[,user@host.sub.domain...]”
Default value: root on the local host
Qmgr: set server managers+=boss@sol.domain.com

`max_array_size`
The maximum number of subjobs (separate indices) that are allowed in an array job. Format: integer. Default value:10000.

`max_running` The maximum number of jobs allowed to be selected for execution at any given time.
Format: integer
Default value: none
Qmgr: set server max_running=24

`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. The named resource can be any valid PBS resource, such as “ncpus”, “mem”, “pmem”, etc. This limit can be specified as either a *hard* or *soft* limit. (See also section 2.5 “Hard and Soft Limits” on page 16.)

Format:
"max_group_res.resource_name=value[,...]"
Format:
"max_group_res_soft.resource_name=value[,...]"
Default value: none
Qmgr: set server max_group_res.ncpus=10
Qmgr: set server max_group_res_soft.mem=1GB

The first line in the example above sets a normal (e.g. *hard*) limit of 10 CPUs as the aggregate maximum that any group may consume. The second line in the example illustrates setting a group *soft* limit of 1GB of memory.

This limit cannot be applied selectively to primetime or non-primetime. Use a cron script to turn this limit on and off for that.

`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.This limit can be specified as either a *hard* or *soft* limit. (See also section 2.5 "Hard and Soft Limits" on page 16.)
Format: integer
Default value: none
Qmgr: set server max_group_run=10
Qmgr: set server max_group_run_soft=7

`max_user_res,`
`max_user_res_soft`

The maximum amount of the specified resource that any single user may consume. The named resource can be any valid PBS resource, such as "ncpus", "mem", "pmem", etc. This limit can be specified as either a *hard* or *soft* limit. (See also section 2.5 "Hard and Soft Limits" on page 16.)
Format: "max_user_res.resource_name=value[,...]"
Format:
"max_user_res_soft.resource_name=value[,...]"

Default value: none
Qmgr: set server max_user_res.ncpus=6
Qmgr: set server max_user_res_soft.ncpus=3

The first line in the example above sets a normal (e.g. *hard*) limit of 6 CPUs as a maximum that any single user may consume. The second line in the example illustrates setting a *soft* limit of 3 CPUs on the same resource.

`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. This limit can be specified as either a *hard* or *soft* limit. (See also section 2.5 "Hard and Soft Limits" on page 16.)
Format: integer
Default value: none
Qmgr: set server max_user_run=6
Qmgr: set server max_user_run_soft=3

`node_fail_requeue`

This server attribute controls how long the server will wait before requeueing or deleting a job when it loses contact with the primary execution host. (If the job is running on more than one execution host and the primary execution host loses contact with a non-primary execution host, the node_fail_requeue attribute does not apply. In this case the job is immediately requeued or deleted.)

See section 2.6.1 "Node Fail Requeue" on page 36.

Requires either Manager or Operator privilege to set.
Format: integer
Default value: 310 (seconds)
Qmgr: set server node_fail_requeue=200

`node_group_enable`

When true directs the Server to enable node grouping. Requires Manager privilege to set or alter. See also `node_group_key`, and section 4.6.12 “Node Grouping” on page 192.
Format: boolean
Default value: disabled
Qmgr: set server node_group_enable=true

`node_group_key`

Specifies the resource to use for node grouping. Must be a string or string_array. Requires Manager privilege to set or alter. See also `node_group_enable`, and section 4.6.12 “Node Grouping” on page 192.
Format: string
Default value: disabled
Qmgr: set server \
node_group_key=*resource[,resource ...]*

`node_pack` Deprecated.

`operators` List of users granted PBS Operator privileges. Format of the list is identical with `managers` above. Requires Manager privilege to set or alter.
Format: “user@host.sub.domain[,user@host.sub.domain...]”
Default value: root on the local host.
Qmgr: set server \
operators+=user1@sol.domain.com
Qmgr: set server operators=user1@*.domain.com
Qmgr: set server operators=user1@*

`pbs_license_file_location`

Hostname of license server, or local pathname to the actual license file(s), which is associated with a license server. String. Set by PBS Manager. Readable by all. Default value: empty string, meaning no server to contact. section 5.4.3.1 “Setting the License File Location in pbs_license_file_location” on page 96 in the PBS Professional Installation & Upgrade Guide.

The ALTAIR_LM_LICENSE_FILE environment variable is set by the server to the same value as this attribute.

To set `pbs_license_file_location` to the hostname of the license server:

qmgr> set server pbs_license_file_location= \
<port1>@<host1>:<port2>@<host2>:…
:<portN>@<hostN>

where <host1>, <host2>, …, <hostN> can be IP addresses.

To set `pbs_license_file_location` to a local path:

qmgr> set server pbs_license_file_location= \
<path_to_local_license_file> \
[[:<path_to_local_license_file2>]:…\
:<path_to_local_license_fileN>]]

To unset `pbs_license_file_location`:

```
Qmgr> unset server \
pbs_license_file_location
```

`pbs_license_linger_time`
The number of seconds to keep an unused CPU license, when the number of licenses is above the value given by `pbs_license_min`. Time. Set by PBS Manager. Readable by all. Default: 3600 seconds. See section 5.4.3.4 “Setting pbs_license_linger_time” on page 99 in the PBS Professional Installation & Upgrade Guide.

To set `pbs_license_linger_time`:

Qmgr> set server \
pbs_license_linger_time=<Z>

To unset `pbs_license_linger_time`:

Qmgr> unset server \
pbs_license_linger_time

`pbs_license_max`

Maximum number of licenses to be checked out at any time, i.e maximum # of CPU licenses to keep in the PBS local license pool. Sets a cap on the number of CPUs that can be licensed at one time. Long. Set by PBS Manager. Readable by all. Default: maximum value for an integer. section 5.4.3.4 “Setting pbs_license_linger_time” on page 99 in the PBS Professional Installation & Upgrade Guide.

To set `pbs_license_max`:

`qmgr> set server \`
pbs_license_max=<Y>

To unset `pbs_license_max`:

`Qmgr> unset server \`
pbs_license_max

`pbs_license_min`

Minimum number of CPUs to permanently keep licensed, i.e. the minimum # of CPU licenses to keep in the PBS local license pool. This is the minimum number of licenses to keep checked out. Long. Set by PBS Manager. Readable by all. Default: zero. section 5.4.3.2 “Setting pbs_license_min” on page 97 in the PBS Professional Installation & Upgrade Guide.

This is for specifying the minimum # of CPU licenses that must be checked-out at any given time. That is, The default value is 0.

To set `pbs_license_min`:

Qmgr> set server pbs_license_min=<X>

To unset `pbs_license_min`:

`Qmgr> unset server \`
pbs_license_min)

`query_other_jobs`
The setting of this attribute controls whether or not general users, other than the job owner, are allowed to query the status of or select the job. Requires Manager privilege to set or alter.
Format: boolean
Default value: true (users may query or select jobs owned by other users)
Qmgr: set server query_other_jobs=false

`resources_available`
List of resources and amounts available to jobs on this Server. The sum of the resources of each type used by all jobs running by this Server cannot exceed the total amount listed here.
Format:
"resources_available.resource_name=value[,...]"
Default value: unset
Qmgr: set server resources_available.ncpus=16
Qmgr: set server resources_available.mem=400mb

`resources_default`
The list of default resource values that are set as limits for a job executing on this Server when the job does not specify a limit, and there is no queue default. The job inherits this list when there is no queue default. The values for `resources_default` are not derived from any

other values; they are either set or not set. See also section 2.11 “Resource Defaults” on page 78.
Format:
“resources_default.resource_name=value[,...]
Default value: for ncpus, the default value is 1
Qmgr: set server resources_default.mem=8mb
Qmgr: set server resources_default.ncpus=1
Qmgr: s s resources_default.place="pack:shared"

`resources_max`

Maximum amount of each resource which can be requested by a single job on this Server if there is not a `resources_max` valued defined for the queue in which the job resides. See section 2.11 “Resource Defaults” on page 78.
Format: “resources_max.resource_name=value[,...]
Default value: infinite usage
Qmgr: set server resources_max.mem=1gb
Qmgr: set server resources_max.ncpus=32

`resv_enable`

This attribute is a master switch to turn on/off advance and standing reservation capability on the Server. If set False, reservations are not accepted by the Server, however any already existing reservations will not be automatically removed. If this attribute is set True the Server will accept reservation requests. See section 4.8 “Advance and Standing Reservations” on page 204. Requires Manager privilege to set or alter.
Format: boolean
Default value: True
Qmgr: set server resv_enable=true

`rpp_highwater`

The maximum number of RPP packets that can be in transit at any time. Acceptable values: Greater than or equal to one. Integer. Default: 64. Settable by Manager. Visible to all.
Qmgr: set server rpp_highwater=100

`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. Integer. Acceptable values: Greater than or equal to zero. Default: 10. Settable by Manager. Visible to all.
Qmgr: set server rpp_retry=12

`scheduler_iteration`
The time, in seconds, between iterations of attempts by the Scheduler to schedule jobs. On each iteration, the Scheduler examines the available resources and runnable jobs to see if a job can be initiated. This examination also occurs whenever a running job terminates or a new job is placed in the queued state in an execution queue.
Format: integer seconds
Default value: 600
Qmgr: set server scheduler_iteration=300

`scheduling` Controls if the Server will request job scheduling by the PBS Scheduler. If true, the Scheduler will be called as required; if false, the Scheduler will not be called and no job will be placed into execution unless the Server is directed to do so by a PBS Operator or Manager. Setting or resetting this attribute to `true` results in an immediate call to the Scheduler. The PBS installation script sets **`scheduling`** to True. However, a call to **`pbs_server -t create`** sets **`scheduling`** to false.
Format: boolean
Default value: value of `-a` option when Server is invoked; if `-a` is not specified, the value is recovered from the prior Server run. Qmgr: set server scheduling=true

`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. An unset value is treated

as `false`. See discussion of use, and caveats, in section section 2.14 “Password Management for Windows” on page 87.
The feature can be enabled (set to True) only if no jobs exist, or if all jobs are of type “p” hold (bad password).
Format: boolean. It can be disabled only if there are no jobs currently in the system.
Default: false (UNIX), true (Windows)

Qmgr: set server \
single_signon_password_enable=true

The following attributes are read-only: they are maintained by the Server and cannot be changed by a client.

`FLicenses` Shows the number of floating PBS licenses currently available for allocation to unlicensed CPUs. One license is required for each virtual CPU. The scheduler uses this is the attribute to determine the number of licenses available.

`license_count`

Count of available licenses. Snapshot taken every 5 minutes. `license_count`= Avail_Global:<X> Avail_Local:<Y> Used:<Z> High_Use:<W>

Avail_Global is the number of PBS CPU licenses still kept by the Altair License Server (checked-in).

Avail_Local is the number of PBS CPU licenses in the internal PBS license pool (checked-out).

Used is the number of PBS CPU licenses currently in use.

High_Use is the highest number of CPU licenses checked-out and used at any given time while the current instance of the PBS server is running.

"Avail_Global" + "Avail_Local" + "Used" is the total number of CPU licenses configured for the PBS complex.

Integer. Set by Server. Readable by all. Default: zero.

`pbs_version` The release version number of the Server.

`resources_assigned`
The total amount of certain resources allocated to running jobs. The resources allocated to a job from vnodes will not be released until certain allocated resources such as cpusets have been freed by all MOMs running the job.

`server_host` The name of the host on which the current (Primary or Secondary) Server is running, in failover mode.

`state_count` Tracks the number of jobs in each state currently managed by the Server

`server_state` The current state of the Server. Possible values are:

Table 5:

Active	The Server is running and will invoke the Scheduler as required to schedule jobs for execution.
Hot_Start	The Server may remain in this state for up to five minutes after being restarted with the "hot" option on the command line. Jobs that are already running will remain in that state and jobs that got requeued on shutdown will be rerun.
Idle	The Server is running but will not invoke the Scheduler.
Scheduling	The Server is running and there is an outstanding request to the Scheduler.

Table 5:

Terminating	The Server is terminating. No additional jobs will be scheduled.
Terminating, Delayed	The Server is terminating in delayed mode. The Server will not run any new jobs and will shut down when the last currently running job completes.

`total_jobs` The total number of jobs currently managed by the Server.

2.6.1 Node Fail Requeue

This server attribute controls how long the server will wait before requeueing or deleting a job when it loses contact with the primary execution host. (If the job is running on more than one execution host and the primary execution host loses contact with a non-primary execution host, the node_fail_requeue attribute does not apply. In this case the job is immediately requeued or deleted.)

Whether a job is requeued or deleted is controlled by its `rerunnable` attribute. If a job's `rerunnable` attribute is set to "y", then the job is requeued. If the job's `rerunnable` attribute is set to "n", the job is deleted. See the "`-r y|n`" option to the `qsub` command in the **PBS Professional User's Guide**.) If a job is deleted, mail is sent to the owner of the job.

The server waits for the specified number of seconds, then attempts to contact the primary execution host, then kills and requeues the job if it cannot contact the host. If the value is zero or is unset, the job is neither killed nor requeued, but allowed to continue running. If the value is negative, it is treated as if it were set to 1 second.

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. When the server loses contact with an execution host, all jobs for which this is the primary execution host are requeued or killed at the same time.

When a job is thus requeued, it retains its original place in its original queue with its former priority. This usually means that it is the next job to be started. Exceptions are when another higher-priority job was submitted after the requeued job started, or when this job's owner is over their fair-share limit.

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.

Once a job is requeued or aborted, the resources allocated to the job cannot be made available until they are actually (a) freed or (b) made shareable to other jobs.

Manager or Operator privilege is required to set this attribute.
Format: integer
Default value: 310 (seconds)
Qmgr: **set server node_fail_requeue=200**

2.7 Queues Within PBS Professional

Once you have the Server attributes set the way you want them, you will next want to review the queue settings. The default (binary) installation creates one queue with the attributes shown in the example below. You may wish to change these settings or add other attributes or add additional queues. The following discussion will be useful in modifying the PBS queue configuration to best meet your specific needs.

2.7.1 Execution and Routing Queues

There are two types of queues defined by PBS: routing and execution. A **routing queue** is a queue used to move jobs to other queues including those which exist on different PBS Servers. 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. In spite of the name, jobs in a queue need not be processed in queue-order (first-come first-served or *FIFO*).

A Server may have multiple queues of either or both types, but there must be at least one queue defined. Typically it will be an execution queue; jobs cannot be executed while residing in a routing queue.

See the following sections for further discussion of execution and route queues:

section 2.7.4 “Attributes of Execution Queues Only” on page 45
section 2.7.5 “Attributes for Route Queues Only” on page 47
section 2.13 “Selective Routing of Jobs into Queues” on page 84
section 2.15.6 “Failover and Route Queues” on page 104
section 8.4 “Complex Multi-level Route Queues” on page 404.

2.7.2 Creating Queues

To create an execution queue exec_queue:

```
Qmgr:
create queue exec_queue
set queue exec_queue queue_type = Execution
set queue exec_queue enabled = true
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
set queue routing_queue queue_type = Route
set queue routing_queue route_destinations =
exec_queue
```

Note:
1. Destination queues must be created before being used as the routing queue’s route_destinations.
2. Routing queue’s route_destinations must be set before enabling and starting the routing queue.

```
set queue routing_queue enabled = true
set queue routing_queue started = true
```

Note:
If we want the destination queue to accept jobs only from a routing queue, we set its from_route_only attribute to true:

```
set queue exec_queue from_route_only = True
```

2.7.3 Queue Configuration Attributes

Queue configuration attributes fall into three groups: those which are applicable to both types of queues, those applicable only to execution queues, and those applicable only to routing queues. If an "execution queue only" attribute is set for a routing queue, or vice versa, it is simply ignored by the system. However, as this situation might indicate the Administrator made a mistake, the Server will issue a warning message (on stderr) about the conflict. The same message will be issued if the queue type is changed and there are attributes that do not apply to the new type.

Queue public attributes are alterable on request by a client. The client must be acting for a user with Manager or Operator privilege. Certain attributes require the user to have full Administrator privilege before they can be modified. The following attributes apply to both queue types:

`acl_group_enable`
: When true directs the Server to use the queue's group access control list `acl_groups`.
Format: boolean
Default value: false = disabled
Qmgr: set queue *QNAME* acl_group_enable=true

`acl_groups`
: List which allows or denies enqueuing of jobs owned by members of the listed groups. The groups in the list are groups on the Server host, not submitting host. Note that the job's execution GID is evaluated (which is either the user's default group, or the group specified by the user via the `-Wgroup_list` option to `qsub`.) See also `acl_group_enable`.
Format: "[+|-]group_name[,...]"
Default value: unset
Qmgr: set queue *QNAME* acl_groups="math,physics"

`acl_host_enable`
When true directs the Server to use the `acl_hosts` access list for the named queue.
Format: boolean
Default value: disabled
Qmgr: set queue *QNAME* acl_host_enable=true

`acl_hosts`
List of hosts which may enqueue jobs in the queue. See also `acl_host_enable`.
Format: “[+|-]hostname[,...]”
Default value: unset
Qmgr: set queue *QNAME* acl_hosts=”sol,star”

`acl_user_enable`
When true directs the Server to use the `acl_users` access list for this queue.
Format: boolean (see acl_group_enable)
Default value: disabled
Qmgr: set queue *QNAME* acl_user_enable=true

`acl_users`
A single list of users allowed or denied the ability to enqueue jobs in this queue. Requires Manager privilege to set or alter. See also `acl_user_enable`. Manager privilege overrides user access restrictions. The order of the elements in the list is important. The list is searched, starting at the beginning, for a match. The first match encountered in the list is accepted and terminates processing. Therefore, to allow all users except for some, the list of denied users should be put at the front of the list, followed by the set of allowed users. When usernames are added to the list, they are appended to the end of the list.
Format: “[+|-]user[@host][,...]”
Default value: all users allowed
To set list of allowed users:
Qmgr: set queue QNAME acl_users=”-bob,-tom,joe,+”
To add to list of allowed users:
Qmgr: set queue QNAME acl_users+=nancy@terra

To remove from list of allowed users:
Qmgr: set queue QNAME acl_users-=joe
To add to list of disallowed users:
Qmgr: set queue QNAME acl_users+=-mary

`enabled`
When true, the queue will accept new jobs. When false, the queue is *disabled* and will not accept jobs.
Format: boolean
Default value: disabled
Qmgr: set queue *QNAME* enabled=true

`from_route_only`
When true, this queue will accept jobs only when being routed by the Server from a local routing queue. This is used to force users to submit jobs into a routing queue used to distribute jobs to other queues based on job resource limits.
Format: boolean
Default value: disabled
Qmgr: set queue *QNAME* from_route_only=true

`max_array_size`
The maximum number of subjobs that a job array in that queue can have. Job arrays with more than this number will be rejected at qsub time.
Format: integer.
Default: 10000.
Qmgr: set queue QNAME max_array_size = 5000

`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. The named resource can be any valid PBS resource, such as "ncpus", "mem", "pmem", etc. This limit can be specified as either a *hard* or *soft* limit. (See also section 2.5 "Hard and Soft Limits" on page 16.)
Format:
"max_group_res.resource_name=value[,...]"

Format:
"max_group_res_soft.resource_name=value[,...]"
Default value: none
Qmgr: set queue *QNAME*
max_group_res.mem=1GB
Qmgr: set queue *QNAME*
max_group_res_soft.ncpus=10

The first line in the example above sets a normal (e.g. *hard*) limit of 1GB on memory as the aggregate maximum that any group in this queue may consume. The second line in the example illustrates setting a group *soft* limit of 10 CPUs.

`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.This limit can be specified as either a *hard* or *soft* limit. (See also section 2.5 "Hard and Soft Limits" on page 16.)
Format: integer
Default value: none
Qmgr: set queue QUEUE max_group_run=10
Qmgr: set queue QUEUE max_group_run_soft=7

`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.
Format: integer
Default value: infinite
Qmgr: set queue *QNAME* max_queuable=200

`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. The named resource can be any valid PBS resource, such as "ncpus", "mem", "pmem", etc. This limit can be specified as either a *hard* or *soft*

limit. (See also section 2.5 “Hard and Soft Limits” on page 16.)
Format: “max_user_res.resource_name=value[,...]”
Format:
“max_user_res_soft.resource_name=value[,...]”
Default value: none
Qmgr: set queue *QNAME* max_user_res.ncpus=6
Qmgr: set queue *QNAME*
max_user_res_soft.ncpus=3

`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. This limit can be specified as either a *hard* or *soft* limit. (See also section 2.5 “Hard and Soft Limits” on page 16.)
Format: integer
Default value: none
Qmgr: set queue QUEUE max_user_run=6
Qmgr: set queue QUEUE max_user_run_soft=3

`node_group_key`
Specifies the resource to use for node grouping. Must be a string or string_array. Overrides server's `node_group_key`. Format: string. Default value: disabled. Example:
Qmgr: set queue Q \
node_group_key=RESOURCE[,RESOURCE ...]

`priority` The priority of this queue against other queues of the same type on this Server. (A larger value is higher priority than a smaller value.) May affect job selection for execution/routing.
Format: integer
Default value: 0
Qmgr: set queue *QNAME* priority=123

`queue_type` The type of the queue: execution or route. This attribute must be explicitly set.
Format: “execution”, “e”, “route”, “r”

Default value: none, must be specified
Qmgr: set queue *QNAME* queue_type=route
Qmgr: set queue *QNAME* queue_type=execution

`resources_default`

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 the queue's **`resources_default`** is 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`. See also section 2.11 "Resource Defaults" on page 78.
Format: "resources_default.*resource_name*=*value*"
Default value: none
Qmgr: set queue *QNAME* resources_default.mem=1kb
Qmgr: set queue *QNAME* resources_default.ncpus=1
Qmgr: set queue QNAME resources_default.place="pack:shared"

`resources_max`

The maximum amount of each resource which can be requested by a single job in this queue. The queue value supersedes any Server wide maximum limit. See also section 2.11 "Resource Defaults" on page 78.
Format: "resources_max.*resource_name*=*value*"
Default value: unset
Qmgr: set queue *QNAME* resources_max.mem=2gb
Qmgr: set queue *QNAME* resources_max.ncpus=32

`resources_min`

The minimum amount of each resource which can be requested by a single job in this queue. See also section 2.11 "Resource Defaults" on page 78.
Format: "resources_min.*resource_name*=*value*"
Default value: unset
Qmgr: set queue *QNAME* resources_min.mem=1kb

Qmgr: set queue *QNAME* resources_min.ncpus=1

`started` When true, jobs may be scheduled for execution from this queue. When false, the queue is considered *stopped* and jobs will not be executed from this queue.
Format: boolean
Default value: unset
Qmgr: set queue *QNAME* started=true

2.7.4 Attributes of Execution Queues Only

`checkpoint_min`
Specifies the minimum interval of CPU time, in minutes, which is allowed between checkpoints of a job. If a user specifies a time less than this value, this value is used instead.
Format: integer
Default value: unset
Qmgr: set queue *QNAME* checkpoint_min=5

`default_chunk`
Defines default elements of chunks for all jobs on this queue. All jobs will inherit default chunk elements for elements not set at submission time, if server and queue resources_default do not apply. See the pbs_resources(7B) man page. Jobs moved to this queue from another queue will lose their old defaults and inherit these.
Format: resource specification format, e.g. "default_chunk.resource=value,default_chunk.resource=value, ..."
Qmgr: `set queue QNAME default_chunk.mem=100mb`

`kill_delay` The amount of the time delay between the sending of SIGTERM and SIGKILL when a `qdel` command is issued against a running job.
Format: integer seconds
Default value: 2 seconds
Qmgr: set queue *QNAME* kill_delay=5

`max_running` The maximum number of jobs allowed to be selected from this queue for routing or execution at any given time. For a routing queue, this is enforced by the Server, if set.
Format: integer
Default value: infinite
Qmgr: set queue *QNAME* max_running=16

`max_user_run` The maximum number of jobs owned by a single user that are allowed to be running from this queue at one time.
Format: integer
Default value: unset
Qmgr: set queue *QNAME* max_user_run=5

`max_group_run`
The maximum number of jobs owned by users in a single group that are allowed to be running from this queue at one time.
Format: integer
Default value: unset
Qmgr: set queue *QNAME* max_group_run=20

`resources_available`
The list of resource 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.
Format:
"resources_available.*resource_name*=*value*"
Default value: unset
Qmgr: set queue *QNAME*
resources_available.mem=1gb

2.7.5 Attributes for Route Queues Only

`route_destinations`

The list of destinations to which jobs may be routed, listed in the order that they should be tried. See also section 2.13 “Selective Routing of Jobs into Queues” on page 84.
Format: queue_name[,...]
Default value: none, should be set to at least one destination.
Qmgr: set queue *QNAME* route_destinations=*QueueTwo*

`route_held_jobs`

If true, jobs with a hold type set may be routed from this queue. If false, held jobs are not to be routed.
Format: boolean
Default value: false = disabled
Qmgr: set queue *QNAME* route_held_jobs=true

`route_lifetime`

The maximum time a job is allowed to exist in a routing queue. If the job cannot be routed in this amount of time, the job is aborted. If unset, the lifetime is infinite.
Format: integer seconds
Default value: infinite
Qmgr: set queue *QNAME* route_lifetime=600

`route_retry_time`

Time delay between route retries. Typically used when the network between servers is down.
Format: integer seconds. Default value: 30
Qmgr: set queue *QNAME* route_retry_time=120

`route_waiting_jobs`

If true, jobs with a future `execution_time` attribute may be routed from this queue. If false, they are not to be routed.
Format: boolean
Default value: false = disabled
Qmgr: set queue *QNAME* route_waiting_jobs=true

2.7.6 Read-only Attributes of Queues

These attributes are visible to client commands, but cannot be changed by them.

`hasnodes` If true, indicates that the queue has vnodes associated with it.

`total_jobs` The number of jobs currently residing in the queue.

`state_count` Lists the number of jobs in each state within the queue.

`resources_assigned`
Amount of resources allocated to jobs running in this queue.

2.7.7 Queue Status

When you use the qstat command to find the status of a queue, it is reported in the “State” field. The field will show two letters. One is either E (enabled) or D (disabled.) The other is R (running, same as started) or S (stopped.)

2.8 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. Commands such as

```
Qmgr: create node VNODE
```

have not changed, and operate on vnodes despite referring to nodes. However, only the natural vnode on a multi-vnode host should be created this way. See the `pbs_node_attributes(7B)` man page.

On Windows, there is a one-to-one correspondence between MOMs and vnodes.

2.8.1 Where Jobs Run

Where jobs will be run is determined by an interaction between the Scheduler and the Server. This interaction is affected by the list of hosts known to the server, and the system configuration onto which you are deploying PBS. Without this list of vnodes, the Server will not establish a communication stream with the MOM(s) and MOM will be unable to report information about running jobs or notify the Server when jobs complete. If the PBS configuration consists of a single host on which the Server and MOM are both running, all the jobs will run there.

If your complex has more than one execution host, then distributing jobs across the various hosts is a matter of the Scheduler determining on which host to place a selected job. By default, when the Scheduler seeks a vnode meeting the requirements of a job, it will select the first available vnode in the list that meets those requirements. Thus the order of vnodes in the nodes file has a direct impact on vnode selection for jobs. (This default behavior can be overridden by the various vnode-sorting options available in the Scheduler. For details, see the discussion of `node_sort_key` in section 4.3 “Scheduler Configuration Parameters” on page 162.)

Use the `qmgr` command to create or delete vnodes. See section 2.8.3 “Creating or Modifying Vnodes” on page 50. Only use the **`qmgr`** command to create or delete vnodes.

Vnodes can have attributes and resources associated with them. Attributes are name=value pairs, and resources use name.resource=value pairs. A user’s job can specify that the vnode(s) used for the job have a certain set of attributes or resources. See section 2.10 “PBS Resources” on page 60.

2.8.2 Natural Vnodes

A natural vnode does not correspond to any actual hardware. It is used to define any placement set information that is invariant for a given host. See section 4.6 “Placement Sets and Task Placement” on page 176. It is defined as follows:

`name` The name of the natural vnode is, by convention, the MOM contact name, which is usually the hostname. The MOM contact name is the vnode's Mom attribute. See the pbs_node_attributes(7B) man page.

`pnames attribute` An attribute, "pnames", with value set to the list of resource names that define the placement sets' types for this machine.

`sharing attribute` An attribute, "sharing" is set to the value "ignore_excl".

The order of the pnames attribute follows placement set organization. If name X appears to the left of name Y in this attribute's value, an entity of type X may be assumed to be smaller (that is, be capable of containing fewer vnodes) than one of type Y. No such guarantee is made for specific instances of the types.

Natural vnodes should not have their 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.

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

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

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.

2.8.3 Creating or Modifying Vnodes

After `pbs_server` is started, the vnode list may be created via the `qmgr` command. First start up pbs_mom, then use qmgr to add the vnode. For example, to add a new vnode, use the “create” sub-command of `qmgr`:

```
create node vnode_name [attribute=value]
```

where the attributes and their associated possible values are shown in the table below. Vnode attributes cannot be used as vnode names. On a multi-vnode system, only the natural vnode should be created this way. Vnode attributes are listed in section 2.9 “Vnode Configuration Attributes” on page 53.

Important: All comma-separated attribute-value strings must be enclosed in quotes.

Below are several examples of creating vnodes via `qmgr`.

```
qmgr
Qmgr: create node mars \
   resources_available.ncpus=2
Qmgr: create node venus
```

Modify vnodes: Once a vnode has been created, its attributes and/or boolean resources can be modified using the following `qmgr` syntax:

```
set node vnode_name \
   [attribute[+|-]=value]
```

where attributes are the same as for `create`. For example:

```
qmgr
Qmgr: set node mars
resources_available.inner=true
Qmgr: set node mars
resources_available.haslife=true
```

Delete vnodes: Nodes can be deleted via `qmgr` as well, using the `delete node` syntax, as the following example shows:

```
qmgr
Qmgr: delete node mars
Qmgr: delete node pluto
```

2.8.3.1 Caveats

Most of a vnode's attributes may be set using qmgr. However, some **must** be set on the individual execution host in local vnode definition files, NOT by using qmgr. Those that must be set on the execution host this way are

sharing
ncpus
mem
vmem

An example of the way to do this (in this case, changing the "sharing" attribute for a vnode named V10) uses the script "change_sharing". See section 3.2.1 "Creation of Site-defined MOM Configuration Files" on page 109.

```
# cat change_sharing
$configversion 2
V10:  sharing = ignore_excl
# . /etc/pbs.conf
# $PBS_EXEC/sbin/pbs_mom -s insert \
ignore_excl change_sharing
# pkill -HUP pbs_mom
```

Do **not** set sharing, ncpus, mem, or vmem on a vnode via qmgr.

It is not a good idea to try to use qmgr to create the vnodes for an Altix, 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. 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. Be default, MOM only knows about the natural vnode, the one whose name is the same as the host.

2.9 Vnode Configuration Attributes

A vnode has the following configuration attributes:

`comment`
: General comment; can be set by a PBS Manager. If this attribute is not explicitly set, the PBS Server will use it to display vnode status, specifically why the vnode is down. If explicitly set by the Administrator, it will not be modified by the Server.
Format: string
Qmgr: set node *MyNode* comment="Down until 5pm"

`lictype`
: Deprecated. No longer used.

`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

`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

`Mom`
: Hostname of host on which MOM daemon will run. Can be explicitly set only via `qmgr`, and only at vnode creation. Defaults to value of vnode resource (vnode name.)

`no_multinode_jobs`
: If this attribute is set true, jobs requesting more than one chunk will not be run on this vnode. This

attribute can be used in conjunction with Cycle Harvesting on workstations to prevent a select set of workstations from being used when a busy workstation might interfere with the execution of jobs that require more than one vnode. It is not recommended to set this attribute on vnodes in a multi-vnoded machine.
Format: boolean
Qmgr: set node *MyNode* no_multinode_jobs=true

`Port` Port number on which MOM will listen. Integer. Can be explicitly set only via `qmgr`, and only at vnode creation. On multi-vnode machine, can only be set on natural vnode.

`priority` The priority of this vnode against other vnodes of the same type on this Server. (A larger value is higher priority than a smaller value.) May be used in conjunction with `node_sort_key`.
Format: integer
Default value: 0
Qmgr: set node *MyNode* priority=123

`queue` Name of an execution queue (if any) associated with a vnode. If this attribute is set, only jobs from the named queue will be run on the associated vnode, and jobs in that queue will only be run on the vnode or vnodes associated with that queue. Note: a vnode can be associated with at most one queue by this method. Note that if a vnode is associated with a queue, it will no longer be considered for advance or standing reservations, or for node grouping.
Format: queue specification
Qmgr: set node *MyNode* queue=*MyQueue*

`resources_available`
List of resources available on vnode. Any valid PBS resources can be specified.
Format: resource list
Qmgr:set node *MyNode*
resources_available.ncpus=2

Qmgr:set node *MyNode* resources_available.*RES*=*xyz*

`resv_enable` Specifies whether or not the vnode can be used for reservations. A vnode that is configured for cycle harvesting should not be used for reservations. Any reservations already assigned to this vnode will not be removed if this attribute is subsequently set to false. Requires manager privilege to set or alter. Format: Boolean. Default value: True.

`sharing` Defines whether more than one job at a time can use this vnode's resources. Either a) the vnode is allocated exclusively to one job, or b) the vnode's unused resources are available to other jobs.
Allowable values: default_shared | default_excl | ignore_excl | force_excl
This attribute can be set via the vnode definition entries in MOM's config file.
Example: `vnodename: sharing=force_excl`
Default value: default_shared.

A vnode's behavior is determined by a combination of its sharing attribute and a job's placement directive. The behavior is defined as follows:

Table 6: Vnode Sharing by Attribute and Placement

Vnode’s sharing attribute	Place Statement Contents		
	unset	place=shared	place=excl
unset	shared	shared	excl
sharing=default_shared	shared	shared	excl
sharing=default_excl	excl	shared	excl
sharing=ignore_excl	shared	shared	shared
sharing=force_excl	excl	excl	excl

The administrator may want to require that each vnode in the system be used exclusively by whatever job is running on it. The administrator should then set "sharing=force_excl". This will override any job "place=shared" setting. Similarly, "sharing=ignore_excl" will override any job "place=excl" setting.

If there is a multi-vnoded system which has a pool of application licenses available for use, these will be associated with a resource defined on the natural vnode (i.e., the vnode whose name is the same as the host). The natural vnode's sharing attribute should be set to "ignore_excl". The pool of licenses will be shared among different jobs. Note that this case does not override a job's "excl" setting. The individual license obtained by the job will be held exclusively. See section 5.7 “Application Licenses” on page 256.

`state` Shows or sets the state of the vnode. Format: string. Qmgr: set node *MyNode* state=offline

Table 7: Node States

State	Set By	Description
free	Server Manager Operator	Node is up and has available CPU(s). Server will mark a vnode “free” on first successful ping after vnode was “down”. Manager/Operator should only use this to clear an “offline” state.
offline	Manager Operator	Node is not usable. Jobs running on this vnode will continue to run. Used by Manager/Operator to mark a vnode not to be used for jobs.
down	Server	Node is not usable. Existing communication lost between Server and MOM.
job-busy	Server	Node is up and all CPUs are allocated to jobs.

Table 7: Node States

State	Set By	Description
job-exclusive	Server	Node is up and has been allocated exclusively to a single job.
busy	Server	Node is up and has load average greater than `$max_load`. When the loadave is above max_load, that node is marked “busy”. The scheduler won’t place jobs on a node marked “busy”. When the loadave drops below ideal_load, the “busy” mark is removed. Consult your OS documentation to determine values that make sense.
stale	Server	MOM managing vnode is not reporting any information. Server can still communicate with MOM.
state-unknown, down	Server	Node is not usable. Since Server’s latest start, no communication with this vnode. May be network or hardware problem, or no MOM on vnode.

A vnode has the following read-only attributes:

`pcpus` Shows the number of physical CPUs on the vnode. On a multi-vnoded machine, this resource will appear only on the first vnode.

`license` Deprecated. Indicates the vnode “license state” as a single character, according to the following table:

Table 8:

u	No jobs are running on this node
f	At least one job has been allocated to this vnode

`ntype` No longer used to distinguish between vnode uses. The “time-shared” and “cluster” node types are deprecated.

`pbs_version` PBS version for the vnode's MOM . Available only to Manager/Operator.

`resources_assigned`
List of resources in use on vnode.
Format: resource list

`reservations` List of reservations pending on the vnode.
Format: reservation specification

`jobs` List of jobs executing on the vnode. A job is listed in the vnode's **`jobs`** attribute until the vnode's resources allocated to the job are freed.

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

2.9.1 Node Comments

Nodes 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
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`. (For details see "The pbsnodes Command" on page 374 and “The xpbsmon GUI Command” on page 396.)

2.9.2 Associating Vnodes with Multiple Queues

You can use resources to associate a vnode with more than one queue. The scheduler will use the resource for scheduling just as it does with any resource. In order to map a vnode to more than one queue, you must define a custom resource. Define the custom resource and add it to the scheduler's sched_priv/sched_config file as follows.

Add to $PBS_HOME/server_priv/resourcedef:

```
Qlist type=string_array flag=h
```

Change $PBS_HOME/sched_priv/sched_config to add "Qlist", e.g.,

```
resources: "ncpus, mem, arch, host, vnode,
Qlist"
```

Now, as an example, assume you have 3 queues: MathQ, PhysicsQ, and ChemQ, and you have 4 vnodes: vn[1], vn[2], vn[3], vn[4]. To achieve the following mapping:

MathQ --> vn[1], vn[2]
PhysicsQ -->vn[2], vn[3], vn[4]
ChemQ --> vn[1], vn[2], vn[3]

Which is the same as:

vn[1] <-- MathQ, ChemQ
vn[2] <-- MathQ, PhysicsQ, ChemQ
vn[3] <-- PhysicsQ, ChemQ
vn[4] <-- PhysicsQ

Set the following via qmgr:

Add queue to vnode mappings:

```
Qmgr: s n vn[1]
resources_available.Qlist="MathQ,ChemQ"
Qmgr: s n vn[2] resources_available.Qlist=
```

```
"MathQ,PhysicsQ,ChemQ"
Qmgr: s n vn[3]
resources_available.Qlist="PhysicsQ,ChemQ"
Qmgr: s n vn[4]
resources_available.Qlist="PhysicsQ"
```

Force jobs to request the correct Q values:

```
Qmgr: s q MathQ resources_default.Qlist=MathQ
Qmgr: s q MathQ resources_min.Qlist=MathQ
Qmgr: s q MathQ resources_max.Qlist=MathQ
Qmgr: s q MathQ default_chunk.Qlist=MathQ

Qmgr: s q PhysicsQ resources_default.Qlist=
PhysicsQ
Qmgr: s q PhysicsQ resources_min.Qlist=
PhysicsQ
Qmgr: s q PhysicsQ resources_max.Qlist=
PhysicsQ
Qmgr: s q PhysicsQ default_chunk.Qlist=
PhysicsQ

Qmgr: s q ChemQ resources_default.Qlist=ChemQ
Qmgr: s q ChemQ resources_min.Qlist=ChemQ
Qmgr: s q ChemQ resources_max.Qlist=ChemQ
Qmgr: s q ChemQ default_chunk.Qlist=ChemQ
```

If you use the vnode's queue attribute, the vnode can be associated only with the queue named in the attribute.

2.10 PBS Resources

Resources can be available on the server and on vnodes. Jobs can request resources. Resources are allocated to jobs, and some resources such as memory are consumed by jobs. The scheduler matches requested resources with available resources, according to rules defined by the administrator. PBS can enforce limits on resource usage by jobs.

PBS provides built-in resources, and in addition, allows the administrator to define custom resources. The administrator can specify which resources are available on a given vnode, as well as at the queue or server level (e.g. floating licenses.) Vnodes can share resources. The administrator can also specify default arguments for `qsub`. These can include resources. See the `qsub(1B)` man page and "Server Configuration Attributes" on page 18.

Resources made available by defining them via resources_available at the queue or server level are only used as job-wide resources. These resources (e.g. `walltime, server_dyn_res`) are requested using -l RESOURCE=VALUE. Resources made available at the host (vnode) level are only used as chunk resources, and can only be requested within chunks using -l select=RESOURCE=VALUE. Resources such as mem and ncpus can only be used at the vnode level in a new-style resource request.

Resources are allocated to jobs both by explicitly requesting them and by applying specified defaults. Jobs explicitly request resources either at the vnode level in chunks defined in a selection statement, or in job-wide resource requests. See the PBS Professional User's Guide and the `pbs_resources(7B)` manual page, "PBS Resources" on page 60 and section 4.3.1 "Rules for Submitting Jobs" on page 42 in the **PBS Professional User's Guide**.

Boolean resources default to "False".

A "consumable" resource is one that is reduced by being used, for example, `ncpus`, licenses, or `mem`. A "non-consumable" resource is not reduced through use, for example, `walltime` or a boolean resource.

Resources are tracked in server, queue, vnode and job attributes. Servers, queues and vnodes have two attributes, resources_available.RESOURCE and resources_assigned.RESOURCE. The resources_available.RESOURCE attribute tracks the total amount of the resource available at that server, queue or vnode, without regard to how much is in use. The resources_assigned.RESOURCE attribute tracks how much of that resource has been assigned to jobs at that server, queue or vnode. Jobs have an attribute called `resources_used.RESOURCE` which tracks the amount of that resource used by that job.

2.10.1 Job Resource Limits

Jobs are assigned limits on the amount of resources they can use. 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.

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.

Job resource limits set a limit for per-job resource usage. Job resource limits are derived in this order from:

explicitly requested job-wide resources (e.g. -l resource=value)
the select specification (e.g. -l select =...)
the queue's default_resources.RES
the server's default_resources.RES
the queue's resources_max.RES
the server's resources_max.RES

The server's default_chunk.RES does **not** affect job-wide limits.
The resources requested for chunks in the select specification are summed, and this sum is used for a job-wide limit. Job resource limits from sums of all chunks override those from job-wide defaults and resource requests.

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.

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.

2.10.2 Unset Resources

When job resource requests are being matched with available resources, a numerical resource that is unset on a host is treated as if it were zero, but 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 is 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 9: Matching Requests to Unset Host-level Resources

Resource Type	Unset Resource	Matching Request Value
boolean	False	False
float	0.0	0.0
long	0	0
size	0	0
string	““““	Never matches
string array	““““	Never matches
time	0, 0:0, 0:0.0, 0:0:0	0, 0:0, 0:0.0, 0:0:0

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* in section 4.3 “Scheduler Configuration Parameters” on page 162.

Note that jobs may be placed on different vnodes from those where they would have run in earlier versions. This is because a job’s resource request will no longer match the same resources on the server, queues and vnodes.

2.10.3 Deleting Custom Resources

If the administrator deletes a resource definition from $PBS_HOME/server_priv/resourcedef and restarts the server, any and all jobs which requested that resource will be purged from the server when it is restarted. Therefore removing any custom resource definition should be done with extreme care.

2.10.4 Vnodes and Shared Resources

Node-level resources can be “ shared” across vnodes. This means that a resource is managed by one vnode, but available for use at others. This is called an *indirect* resource. Any vnode-level dynamic resources (i.e. those listed in the PBS_HOME/sched_priv/sched_config “mom_resources” line) will be treated as “ shared” resources. The MOM manages the sharing. The resource to be shared is defined as usual on the managing vnode. The built-in resource `ncpus` cannot be shared. Static resources can be made indirect.

To set a static value:

```
Qmgr: s n managing_vnode
resources_available.RES =<value>
```

To set a dynamic value, in MOM config:

```
managing_vnode:RES=<value>
managing_vnode:"RES=!path-to-command"
```

To set a “ shared” resource RES on a borrowing vnode, use either

```
Qmgr: s n borrowing_vnode
resources_available.RES=@managing_vnode
```

or in MOM config, for static or dynamic:

```
borrowing_vnode:RES=@managing_vnode
```

Example: to make a static host-level license `dyna-license` on hostA indirect at vnodes hostA0 and hostA1:

```
Qmgr: set node hostA0
resources_available.dyna-license=@hostA
Qmgr: set node hostA1
resources_available.dyna-license=@hostA
```

For example, 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
...

2.10.4.1 Defining 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 to be 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.

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:

```
Qmgr: set node <natural vnode name>
resources_available.ncpus=0
Qmgr: set node <natural vnode name>
resources_available.mem=0
Qmgr: set node <natural vnode name>
resources_available.vmem=0
```

2.10.5 Matching Jobs to Resources

For all resources except boolean and string and string array resources, if a resource is unset (not defined) at a vnode, a resource request will behave as if that resource is zero. If a resource is unset at the server or queue level, the resource request will behave as if that resource is infinite. An unset string or string resource cannot be matched.

For boolean resources, if a resource is unset (undefined) at a server, queue, or vnode, the resource request will behave as if that resource is set to "false". It will match a resource request for that boolean with a value of "false", but not "true".

2.10.6 String Arrays: Multi-valued String resources

The resource of type string_array is a comma-delimited set of strings. Each vnode can have its resource RES be a different set of strings. A job can only request one string per resource in its resource request. The job is placed on a vnode where its requested string is one of the multiple strings set on a vnode.

Example:

- 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

```
Qmgr> set node n4
resources_available.foo_arr+=f7
```

- Vnode n4 now has 4 values of `foo_arr`: f1, f3, f5 and f7

- Submission:

```
qsub –l select=1:ncpus=1:foo_arr=f3
```

A string array resource with one value works exactly like a string resource. A string array uses the same flags as other non-consumable resources. The default value for a job's multi-valued string resource, listed in resource_default.RES, can only be one string.

For string_array resources on a queue, `resources_min` and `resources_max` must be set to the same set of values. A job must request one of the values in the set to be allowed into the queue. For example, if we 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.

2.10.7 Resource Types

The resource values are specified using the following data types:

boolean — Boolean-valued resource. Should be defined only at the vnode level. Non-consumable. Can only be requested inside a select statement, i.e. in a chunk. Name of resource is a string. Allowable values (case insensitive): True|T|Y|1|False|F|N|0

A boolean resource named "RESOURCE" is defined in PBS_HOME/server_priv/resourcedef by putting in a line of the form:

RESOURCE type=boolean flag=h

float — Float. Allowable values: [+-] 0-9 [[0-9] ...][.][[0-9] ...]

long — Long integer. Allowable values: 0-9 [[0-9] ...]

size — Number of bytes (default) or words. It is expressed in the form `integer[suffix]`. The suffix is a multiplier defined in the following table. The size of a word is the word size on the execution host.

Table 10:

b or w	bytes or words.
kb or kw	Kilo (2^{10}, 1024) bytes or words.
mb or mw	Mega (2^{20}, 1,048,576) bytes or words.
gb or gw	Giga (2^{30}, 1,073,741,824) bytes or words.
tb or tw	Tera (2^{40}, or 1024 gigabytes) bytes or words.

string — String. Non-consumable. Allowable values: Any printable character, including the space character., except the tab or other white space and the ampersand (“&”) character. The first character must be alphanumeric or underscore. Only one of the two types of quote characters, " or ', may appear in any given value.

Values:[_a-zA-Z0-9][[-_a-zA-Z0-9 ! " # $ % ´ ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ‘ { | } ~] ...]

string_array — Comma-separated list of strings. Strings in string arrays may not contain commas. Non-consumable. Resource request will succeed if request matches one of the values. Resource request can contain only one string.

time — specifies a maximum time period the resource can be used. Time is expressed in seconds as an integer, or in the form:

```
[[hours:]minutes:]seconds[.milliseconds]
```

Different resources are available on different systems, often depending on the architecture of the computer itself. The table below lists the available resources that can be requested by PBS jobs on any system.

2.10.8 Resource Flags for Consumable or Host-level Resources

Resource flags are added to resource definitions in $PBS_HOME/server_priv/resourcedef. `FLAGS` is a set of characters which indicate whether and how the Server should accumulate the requested amounts of the resource in the attribute `resources_assigned` when the job is run. This allows the server to keep track of how much of the resource has been used, and how much is available.

For example, when defining a static consumable host-level resource, such as a node-locked license, you would use the “n” and “h” flags. However, when defining a dynamic resource such as a floating license, no flag would be used.

The value of `flag` is a concatenation of one or more of the following letters:

`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.
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.

`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.

(none of h, n, f, or q) Indicates a queue-level or server-level resource that is not consumable.

`q` The amount is consumable at the Queue and Server level. Must be consumable or time-based.

Table 11: When to Use Flags

Resource	Server	Queue	Host
Static, consumable	flags = q	flags = q	flags = nh or fh
Static, not consumable	flags = (none of h, n, q or f)	flags = (none of h, n, q or f)	flags = h
Dynamic	(server_dyn_res line in sched_config) flags = (none of h, n, q or f)	(cannot be used)	(MOM config and mom_resources line in sched_config) flags = h

2.10.9 Resource Flags for Resource Permissions

When you are defining a custom resource, you can specify whether users can view or request it, and whether users can qalter a request for that resource. There are two resource permission flags that you can set in the resource definition in $PBS_HOME/server_priv/resourcedef.

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.

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 level and printed to standard error.

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.

While users cannot request these resources, their jobs can inherit default resources from resources_default.RES and default_chunk.RES.

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
```

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's value, the manager must submit the job without requesting the resource, then qalter the resource value.

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`.

2.10.9.1 Command-line Interfaces Affected by Resource Permissions

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.

2.10.10 Built-in Resources

Resource	Description
arch	System architecture. Can be requested only inside of a select statement. One architecture can be defined for a vnode. One architecture can be requested per vnode. Allowable values and effect on job placement are site-dependent. Type: string.
cput	Amount of CPU time used by the job for all processes on all vnodes. Establishes a job resource limit. Can be requested only outside of a select statement. Non-consumable. Type: time.
file	Size of any single file that may be created by the job. Can be requested only outside of a select statement. Type: size.
host	Name of execution host. Can be requested only inside of a select statement. Automatically set to the short form of the hostname in the Mom attribute. Cannot be changed. Site-dependent. Type: string.
mem	Amount of physical memory i.e. workingset allocated to the job, either job-wide or vnode-level. Can be requested only inside of a select statement. Consumable. Type: size.
mpiprocs	Number of MPI processes for this chunk. Defaults to 1 if ncpus > 0, 0 otherwise. Can be requested only inside of a select statement. Type: integer. The number of lines in PBS_NODEFILE is the sum of the values of mpiprocs for all chunks requested by the job. For each chunk with mpiprocs=P, the host name for that chunk is written to the PBS_NODEFILE P times.
mpparch	MPP compute node system type. Can be requested only outside of a select statement. Allowable values: XT or X2. Type: string.

Resource	Description
mppdepth	Depth (number of threads) of each processor. Specifies the number of processors that each processing element will use. Can be requested only outside of a select statement. Default: 1. Type: integer.
mpphost	MPP host. Can be requested only outside of a select statement. Type: string.
mpplabels	List of node labels. Runs the application only on those nodes with the specified labels. Format: comma-separated list of labels and/or a range of labels. Any lists containing commas should be enclosed in quotes escaped by backslashes. For example: `#PBS -l mpplabels=\"red,blue\"` or **`qsub -l mpplabels=\"red,blue\"`** Can be requested only outside of a select statement. Type: string.
mppmem	The maximum memory for all applications. The per-processing-element maximum resident set size memory limit. Can be requested only outside of a select statement. Type: size.
mppnodes	Manual placement list consisting of a comma-separated list of nodes (node1,node2), a range of nodes (node1-node2), or a combination of both formats. Node values are expressed as decimal numbers. The first number in a range must be less than the second number (i.e., 8-6 is invalid). A complete node list is required. Any lists containing commas should be enclosed in quotes escaped by backslashes. For example: `#PBS -l mppnodes=\"40-48,52-60,84,86,88,90\"` or **`qsub -l mppnodes=\"40-48,52-60,84,86,88,90\"`** Can be requested only outside of a select statement. Type: integer.

Resource	Description
mppnppn	Number of processing elements (PEs) per node. Can be requested only outside of a select statement. Type: integer.
mppwidth	Number of processing elements (PEs) for the job. Can be requested only outside of a select statement. Type: integer.
ncpus	Number of processors requested. Cannot be shared across vnodes. Can be requested only inside of a select statement. Consumable. Type: integer.
nice	Nice value under which the job is to be run. Host-dependent. Can be requested only outside of a select statement. Type: integer.
nodect	Deprecated. Number of chunks in resource request from selection directive, or number of hosts requested from node specification. Otherwise defaults to value of 1. Can be requested only outside of a select statement. Read-only. Type: integer.
ompthreads	Number of OpenMP threads for this chunk. Defaults to ncpus if not specified. Can be requested only inside of a select statement. Type: integer. For the MPI process with rank 0, the environment variables NCPUS and OMP_NUM_THREADS are set to the value of ompthreads. For other MPI processes, behavior is dependent on MPI implementation.
pcput	Amount of CPU time allocated to any single process in the job. Establishes a job resource limit. Non-consumable. Can be requested only outside of a select statement. Type: time.
pmem	Amount of physical memory (workingset) for use by any single process of the job. Establishes a job resource limit. Can be requested only outside of a select statement. Consumable. Type: size

Resource	Description
pvmem	Amount of virtual memory for use by the job. Establishes a job resource limit. Can be requested only outside of a select statement. Not consumable. Type: size.
software	Site-specific software specification. Can be requested only outside of a select statement. Allowable values and effect on job placement are site-dependent. Type: string.
vmem	Amount of virtual memory for use by all concurrent processes in the job. Establishes a per-chunk limit. Can be requested only inside of a select statement. Consumable. Type: size.
vnode	Name of virtual node (vnode) on which to execute. For use inside chunks only. Site-dependent. Can be requested only inside of a select statement. Type: string. See the pbs_node_attributes(7B) man page.
walltime	Actual elapsed (wall-clock, except during Daylight Savings transitions) time during which the job can run. Establishes a job resource limit. Can be requested only outside of a select statement. Non-consumable. Default: 5 years. Type: time.

Every consumable resource such as `mem` has four associated values, each of which is used in several places in PBS:

Table 12: Values Associated with Consumable Resources

Value	Node	Queue	Server	Accounting Log	Job	Scheduler
resources_available	X	X	X			X
resources_assigned	X	X	X	X		
resources_used				X	X	X
Resource_List					X	X

The Vnode, Server, and Queue values are usually displayed via pbsnodes and qmgr; the Accounting values appear in the PBS accounting file; and the job values are usually viewed via qstat. The Scheduler values implicitly appear in the Scheduler's configuration file.

The resources_assigned values are reported differently for Vnodes (or Queues, or the Server) versus in the Accounting records. The value of resources_assigned reported for Vnodes (or Queues, or the Server) is the amount directly requested by jobs in the job's Resource_List (without regard to "excl"). The value of the job's resource_assigned (note the singular "resource") reported in the Accounting records is the actual amount assigned to the job by PBS (taking "excl" into account). The job's resource_assigned is not a job attribute. All allocated consumable resources will be included in the "resource_assigned" entries, one resource per entry. Consumable resources include ncpus, mem and vmem by default, and any custom resource defined with the -n or -f flags. A resource will not be listed if the job does not request it directly or inherit it by default from queue or server settings. For example, if a job requests one CPU on an Altix that has four 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.

2.10.10.1 Specifying Architectures

The resources_available.arch resource is the value reported by MOM unless explicitly set by the Administrator. The values for arch are:

Table 13: Values for resources_available.arch

OS	Resource Label
AIX 5	aix4
HP-UX 11	hpux11
Linux	linux
Linux with cpusets	linux_cpuset
Solaris	solaris7
Unicos	unicos
Unicos MK2	unicosmk2

Table 13: Values for resources_available.arch

OS	Resource Label
Unicos SMP	unicossmp

2.10.11 Setting Chunk Defaults

It is possible to set defaults on queues and the Server for resources used within a chunk. For example, the administrator could set the default for ncpus for chunks at the server. This means that if a job requests a certain chunk in which only mem and arch are defined, the default for ncpus will be added to that chunk.

Set the defaults for the server:

```
qmgr
Qmgr: set server  default_chunk.ncpus=1
Qmgr: set server  default_chunk.mem=1gb
```

Set the defaults for queue small:

```
qmgr
Qmgr: set queue small default_chunk.ncpus=1
Qmgr: set queue small default_chunk.mem=512mb
```

2.10.12 Defining New Resources

It is possible for the PBS Manager to define new resources within PBS Professional. Jobs may request these new resources and the Scheduler can be directed to consider the new resources in the scheduling policy. For detailed discussion of this capability, see Chapter 9, "Customizing PBS Resources" on page 237.

2.11 Resource Defaults

The administrator can specify default resources on the server and queue. These resources can be job-wide, which is the same as adding -l RESOURCE to the job's resource request, or they can be chunk resources, which is the same as adding :RESOURCE=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. The administrator can also specify default resources to be added to any `qsub` arguments. In addition, the administrator can specify default placement of jobs.

For example, to set the default architecture on the server:

```
Qmgr: set server resources_default.arch=linux
```

To set default values for chunks, see section 2.10.11 “Setting Chunk Defaults” on page 78.

To set the default job placement for a queue:

```
Qmgr: set queue QUEUE
resources_default.place=free
```

See the PBS Professional User’s Guide for detailed information about how -l `place` is used.

To set the default rerunnable option in a job’s resource request:

```
Qmgr: set server default_qsub_arguments=”-r
y”
```

Or to set a default boolean in a job’s resource request so that jobs don’t run on Red:

```
Qmgr: set server default_qsub_arguments=”-l
Red=false”
```

To set default placement involving a colon:

```
Qmgr: set server
resources_default.place="pack:shared"
```

2.11.1 Jobs and Default Resources

Jobs get default resources, job-wide or per- chunk, with the following order of precedence.

Table 14: Order in which default resources are assigned to jobs

Order of assignment	Default value	Affects Chunks?	Job-wide?
1	Default qsub arguments	If specified	If specified
2	Queue’s default_chunk	Yes	No
3	Server’s default_chunk	Yes	No
4	Queue’s resources_default	No	Yes
5	Server’s resources_default	No	Yes
6	Queue’s resources_max	No	Yes
7	Server’s resources_max	No	Yes

See the `qmgr(8B)` man page for how to set these defaults.

For each chunk in the job's selection statement, first 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 called "default_chunk.RESOURCE".

For example, if the queue in which the job is enqueued has the following defaults defined:

```
default_chunk.ncpus=1
default_chunk.mem=2gb
```

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.`

The job-wide resource request is checked against queue resource defaults, then against server resource defaults. If a default resource is defined which is not specified in the resource request, it is added to the resource request.

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 2.10.9 “Resource Flags for Resource Permissions” on page 71.

2.11.2 The Job’s Resource_List Attribute

The job’s Resource_List attribute lists all the resources requested by the job. These resources come from the job’s resource request and from default resources. Resources requested at the job-wide level are listed as requested, and resources requested within chunks are summed, and the sum is listed. If the job inherited default resources from the server or queue, those are included. Host-level resources are not job-wide and are not included in the job’s Resource_List attribute.

2.11.2.1 Moving Jobs Between Queues

If the job is moved from the current queue to a new queue, any default resources in the job’s resource list inherited from the queue are removed. 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 (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.
Example: 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:

`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

`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

`qsub -lmem=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

`qsub -l nodes=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: `select=4:ncpus=1:mem=1gb`
(In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)

`qsub -l mem=16gb -l nodes=4`

In QA: `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: `select=4:ncpus=1:mem=4gb`

(In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)

2.12 Server and Queue Resource Min/Max Attributes

Minimum and maximum queue and Server limits work with numeric valued resources, including time and size values. Generally, they do not work with string valued resources because of character comparison order. However, setting the `min` and `max` to the same value to force an exact match will work even for string valued resources, as the following example shows.

```
qmgr
Qmgr: set queue big
resources_max.arch=unicos8
Qmgr: set queue big
resources_min.arch=unicos8
```

The above example can be used to limit jobs entering queue `big` to those specifying `arch=unicos8`. Again, remember that if `arch` is not specified by the job, the tests pass automatically and the job will be accepted into the queue.

Note however that if a job does not request a specific resource and is not assigned that resource through default qsub arguments, then the enforcement of the corresponding limit will not occur. To prevent such cases, the Administrator is advised to set queue and/or server defaults. The following example sets a maximum limit on the amount of cputime to 24 hours; but it also has a default of 1 hour, to catch any jobs that do not specify a `cput` resource request.

```
qmgr
Qmgr: set queue big
resources_max.cput=24:00:00
Qmgr: set queue big
resources_default.cput=1:00:00
```

With this configuration, any job that requests more than 24 hours will be rejected. Any job requesting 24 hours or less will be accepted, but will have this limit enforced. And any job that does not specify a `cput` request will receive a default of 1 hour, which will also be enforced.

If a job is submitted without a request for a specific resource, and that resource is specified in the server or queue resources_max, the job may inherit that value for that resource. Whether the job inherits the value in resources_max is determined by the order of inheritance given in section 2.11.1 “Jobs and Default Resources” on page 80.

2.13 Selective Routing of Jobs into Queues

You may want to route jobs to various queues on a Server, or even between Servers, based on the resource requirements of the jobs. The queue attributes *resources_min* and *resources_max* discussed allow this selective routing. The queue’s resources_min/max can only be used with job-wide resources. You cannot use custom host-level resources with queue resources_min/max. This would include any custom resources created with flag=h. That is, you cannot use a custom resource defined with flag=h.

Jobs can only be routed based on resources outside of the select specification, or based on sums of nodal resources.

If you want to use a boolean resource to route jobs w/resources_min/max you will have to define it at the server or queue level (without flag=h.) It will have to be requested with "-l select=x -l <boolean resource>=True". A server or queue-level resource cannot be used to direct a job to an execution node.

As an example, let us assume you wish to establish two execution queues, one for short jobs of less than one minute CPU time, and the other for long running jobs of one minute or longer. Let’s call them `short` and `long`. Apply the `resources_min` and `resources_max` attribute as follows:

```
qmgr
Qmgr: set queue short resources_max.cput=59
Qmgr: set queue long resources_min.cput=60
```

When a job is being enqueued, its requested resource list is tested against the queue limits: `resources_min` <= job_requirement <= `resources_max`. If the resource test fails, the job is not accepted into the queue. Hence, a job asking for 20 seconds of CPU time would be accepted into queue `short` but not into queue `long`.

Important: Note, if the `min` and `max` limits are equal, only that exact value will pass the test.

You may wish to set up a routing queue to direct jobs into the queues with resource limits. For example:

```
qmgr
Qmgr: create queue funnel queue_type=route
Qmgr: set queue funnel route_destinations
="short,long"
Qmgr: set server default_queue=funnel
```

A job will end up in either `short` or `long` depending on its CPU time request.

Important: You should always list the destination queues in order of the most restrictive first as the first queue which meets the job's requirements will be its destination (assuming that queue is enabled).

Extending the above example to three queues:

```
qmgr
Qmgr: set queue short resources_max.cput=59
Qmgr: set queue long resources_min.cput=1:00
Qmgr: set queue long
resources_max.cput=1:00:00
Qmgr: create queue huge queue_type=execution
Qmgr: set queue funnel
route_destinations="short,long,huge"
Qmgr: set server default_queue=funnel
```

A job asking for 20 minutes (20:00) of cup time will be placed into queue `long`. A job asking for 1 hour and 10 minutes (1:10:00) will end up in queue `huge`, because it was not accepted into the first two queues, and nothing prevented it from being accepted into `huge`.

Important: If a test is being made on a resource as shown with `cput` above, and a job does not specify that resource item, and it is not given the resource through defaults, (it does not appear in the `-l resource=`*`valuelist`* on the `qsub` command, the test will pass. In the above case, a job without a CPU time limit will be allowed into queue `short`. You may wish to add a default value to the queues or to the Server.

qmgr
Qmgr: set queue short resources_default.cput=40
or
Qmgr: set server resources_default.cput=40

Either of these examples will ensure that a job without a cup time specification is limited to 40 seconds. A `resources_default` attribute at a queue level only applies to jobs in that queue.

The check for admission of a job to a queue has the following sequence:

1 Clear the job's current defaults (from both existing queue and server)

2 Set new defaults based on named destination queue

3 Test limits against queue min/max and server min/max

4 Clear the job's new defaults

5 Reset the defaults based on the actual queue in which the job resides

If a queue resource default value is assigned, it is done so after the tests against `min` and `max`. Default values assigned to a job from a queue `resources_default` are not carried with the job if the job moves to another queue. Those resource limits become unset as when the job was specified. If the new queue specifies default values, those values are assigned to the job while it is in the new queue. Server level default values are applied if there is no queue level default.

If the job is to be moved into a different queue, then the default values are again cleared and reset based on that destination queue. This happens as the job is enqueued.

If a resource is not set on job submission, it is not checked against the queue's min/max. If no default was set, it won't be included in the Resource_List. The resources_min/max are only checked against equivalent entries in the job's Resource_List. Only consumable resources (those with flag=n or q) are taken from the select specification and turned into separate entries in the Resource_List.

2.13.1 Checks Performed When Jobs are Admitted Into Queues

When a job is being considered for a queue because it was submitted or it was qmoved, the following checks are performed:

Step 1 Any current defaults, either from the server or the current queue, are cleared.

Step 2 New defaults, based on the potential destination queue, are set.

Step 3 The job's limits are tested against the queue and server minima/maxima.

Step 4 The new defaults are cleared.

Step 5 Final defaults are set based on which queue the job was actually enqueued in.

2.14 Password Management for Windows

PBS Professional will allow users to specify two kinds of passwords: a per-user/per-server password, or a per-job password. The PBS administrator must choose which method is to be used. (Discussion of the difference between these two methods is given below; detailed usage instructions for both are given in the **PBS Professional User's Guide**.)

This feature is intended for Windows environments. It should *not* be enabled in UNIX since this feature requires the PBS_DES_CRED feature, which is not enabled in the normal binary UNIX version of PBS Professional. Setting this attribute to "true" in UNIX may cause users to be unable to submit jobs.

The per-user/per-server password was introduced as part of the single signon password scheme. The purpose is to allow a user to specify a password only once and have PBS remember this password to run the user's current and future jobs. A per-user/per-server password is specified by using the command:

```
pbs_password
```

The user must run this command before submitting jobs to the Server. The Server must have the `single_signon_password_enable` attribute set to "true".

Alternatively, one can configure PBS to use the current per-job password scheme. To do this, the Server configuration attribute `single_signon_password_enable` must be set to "false", and jobs must be submitted using:

qsub -Wpwd

You cannot mix the two schemes; PBS will not allow submission of jobs using `-Wpwd` when `single_signon_password_enable` is set to "true".

Important: If you wish to migrate from an older version of PBS Professional on Windows to the current version, be sure to review Chapter 5 of this document, as well as the discussion of `pbs_migrate_users` in Chapter 11.

2.14.1 Single Signon and the qmove Command

A job can be moved (via the `qmove` command) from a Server at hostA to a Server at hostB. If the Server on hostB has single_signon_password_enable set to true, then the user at hostB must have an associated per-user/per-server password. This requires that the user run pbs_password at least once on hostB.

2.14.2 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>
```

2.14.3 Single Signon and Peer Scheduling

In a peer scheduling environment, the Scheduler may move jobs from complex A to complex B. If the Server in complex B has single_signon_password_enable attribute set to true, then users with jobs on complex A must make sure they have per- user/per-server passwords on complex B. This is done by issuing a pbs_password command on complex B.

2.15 Configuring PBS Redundancy and Failover

The redundancy-failover feature of PBS Professional provides the capability for a backup Server to assume the workload of a failed Server, thus eliminating the one single point of failure in PBS Professional. If the Primary Server fails due to a hardware or software error, the Secondary Server will take over the workload and communications automatically. No work is lost in the transition of control.

The following terms are used in this manual section: *Active Server* is the currently running PBS Professional Server process. *Primary Server* refers to the Server process which under normal circumstances is the active Server. *Secondary Server* is a Server which is inactive (idle) but which will become active if the Primary Server fails.

The server attribute values for pbs_license_file_location, `pbs_license_min`, `pbs_license_max`, and `pbs_license_linger_time` are set through the primary server. Since these values are saved in PBS_HOME/server_priv/serverdb, and PBS_HOME is in a shared location, the secondary server can use these licensing parameters. No additional licensing steps are needed for the secondary server to work properly.

2.15.1 Failover Requirements

The following requirements must be met to provide a reliable failover service:

1. The Primary and Secondary Servers must be run on different hosts. Only one Secondary Server is permitted.

2. The Primary and Secondary Server hosts must be the same architecture, i.e. binary compatible, including word length, byte order and padding within the structures.

3. Both the Primary and Secondary Server host must be able to communicate over the network with all execution hosts where a `pbs_mom` is running.

4. The directory and subdirectories used by the Server, `PBS_HOME`, must be on a file system which is available to both the Primary and Secondary Servers. The directory must be readable and writable by root on UNIX, or have Full Control permissions for the local "Administrators" group on the local host on Windows.

   When selecting the failover device, consider both the hardware and the available file systems, as the solution needs to support concurrent read and write access from two hosts. The best solution is a high availability file server device connected to both the Primary and Secondary Server hosts, used in conjunction with a file system that supports both multiple export/mounting and simultaneous read/write access from two or more hosts (such as SGI CXFS, IBM GPFS, or Red Hat GFS).

To avoid introducing a single point of failure, use an NFS file server with the file system exported to and *hard mounted* by both the Primary and Secondary Server hosts. Make sure that neither server host is the machine on which the PBS_HOME file system resides.

In a Microsoft Windows environment, a workable solution is to use the network share facility; that is, use as `PBS_HOME` a directory on a remote Windows host that is shared among primary and secondary server hosts.

5. The /etc/hosts files on the two servers must be set up so that each can find the other and all the hosts in the complex.

Important: Note that a failure of the NFS server will prevent PBS from being able to continue.

6. A MOM, `pbs_mom`, may run on either the Primary or the Secondary hosts, or both, however, this is not recommended. It is strongly recommended that the directory used for "`mom_priv`" be on a local, non-shared, file system. It is critical that the two MOMs do not share the same directory. This can be accomplished by using the `-d` option when starting `pbs_mom`, or with the `PBS_MOM_HOME` entry in the `pbs.conf` file. The `PBS_MOM_HOME` entry specifies a directory which has the following contents:

UNIX:

Table 15:

Directory Contents	Description
aux	Directory with permission 0755
checkpoint	Directory with permission 0700
mom_logs	Directory with permission 0755
mom_priv	Directory with permission 0755
mom_priv/jobs	Subdirectory with permission 0755
mom_priv/config	File with permission 0644
pbs_environment	File with permission 0644
spool	Directory with permission 1777 (drwxrwxrwt)
undelivered	Directory with permission 1777 (drwxrwxrwt)

Windows:

Note: In the table below, references to “access to Admin-account” refer to access to the local Administrators group on the local host.

.

Table 16:

Directory Contents	Description
auxiliary	Directory with full access to *Admin-account* and read-only access to Everyone
checkpoint	Directory with full access only to *Admin-account*
mom_logs	Directory with full access to *Admin-account* and read-only access to Everyone

Table 16:

Directory Contents	Description
mom_priv	Directory with full access to *Admin-account* and read-only access to Everyone
mom_priv/jobs	Subdirectory with full access to *Admin-account* and read-only access to Everyone
mom_priv/config	File with full access-only to *Admin-account*
pbs_environment	File with full access to *Admin-account* and read-only to Everyone
spool	Directory with full access to Everyone
undelivered	Directory with full access to Everyone

If `PBS_MOM_HOME` is present in the `pbs.conf` file, `pbs_mom` will use that directory for its "home" instead of `PBS_HOME`.

7. The version of the PBS Professional commands installed everywhere must match the version of the Server, in order to provide for automatic switching in case of failover.

2.15.2 Failover Configuration for UNIX/Linux

The steps below outline the process for general failover setup, and should be sufficient for configuration under UNIX. To configure PBS Professional for failover operation, follow these steps:

1. Select two systems of the same architecture to be the Primary and Secondary Server systems. They should be binary compatible.

2. Configure a file system (or at least a directory) that is read/write accessible by root (UNIX) from both systems. If an NFS file system is used, it must be "hard mounted" (UNIX) and root or Administrator

must have access to read and write as “root” or as “Administrators” on both systems. Beware of dependencies on remote file systems: 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.

Under Unix, the directory tree must meet the security requirements of PBS. Each parent directory above `PBS_HOME` must be owned by “root” (“Administrators”) and be writable only by “root” (Administrators”).

The NFS lock daemon, lockd, must be running for the file system on the primary and secondary hosts.

3. Install PBS Professional on both systems, specifying the shared file system location for the `PBS_HOME` directory. DO NOT START ANY PBS DAEMONS.

4. Modify /etc/`pbs.conf` file on both systems, as follows:

5. Change `PBS_SERVER` on both systems to the short form of the Primary Server’s hostname. The value must be a valid hostname. Example:

    PBS_SERVER=servername

6. Add the following entries to both `pbs.conf` files; they must have the same value in both files:

    PBS_PRIMARY=primaryname.domain.com
    PBS_SECONDARY= \
    secondaryname.domain.com

where “primaryname.domain.com” is the fully qualified host name of the Primary Server’s host, and “secondaryname.domain.com” is the fully qualified host name of the Secondary Server's host. It is important that these entries be correct and distinct as they are used by the Servers to determine their status at startup.

A sample /etc/pbs.conf file for each server:

Primary:

```
PBS_START_SERVER=1
PBS_START_MOM=0
PBS_START_SCHED=1
PBS_SERVER=primaryname.domain.com
PBS_PRIMARY=primaryname.domain.com
PBS_SECONDARY= \
secondaryname.domain.com
```

Secondary:

```
PBS_START_SERVER=1
PBS_START_MOM=0
PBS_START_SCHED=0
PBS_SERVER=primaryname.domain.com
PBS_PRIMARY=primaryname.domain.com
PBS_SECONDARY= \
secondaryname.domain.com
```

7. These entries must also be added to the `pbs.conf` file on any system on which the PBS commands are installed, and on all execution hosts in the complex.

8. Ensure that the `PBS_HOME` entry on both systems names the shared PBS directory, using the specific path on that host.

9. On the Secondary host, modify the `pbs.conf` file to not start the Scheduler by setting

   PBS_START_SCHED=0

   If needed, the Secondary Server will start a Scheduler itself.

10. It is not recommended to run pbs_mom on both the Primary and Secondary Servers hosts. If you do run a `pbs_mom` on both the Primary and Secondary Server hosts, make sure that `/etc/pbs.conf` on each host has a `PBS_MOM_HOME` defined. This will be local to that host. You will need to replicate the `PBS_MOM_HOME` directory structure at the place specified by `PBS_MOM_HOME`.

11. PBS has a standard delay time from detection of possible Primary Server failure until the Secondary Server takes over. This is discussed in more detail in the “Normal Operation” section below. If your network is very reliable, you may wish to decrease this delay. If your network is unreliable, you may wish to increase this delay. The default delay is 30 seconds. To change the delay, use the “`-F seconds`” option on the Secondary Server's command line:

    pbs_server -F <delay>

12. The Scheduler, `pbs_sched`, is run on the same host as the PBS Server. The Secondary Server will start a Scheduler on its (secondary) host only if the Secondary Server cannot contact the Scheduler on the primary host. This is handled automatically; see the discussion under “Normal Operation” section below.

13. Start up the primary and secondary servers in any order.

14. Once the Primary Server is started, use the `qmgr` command to set or modify the Server's “`mail_from`” attribute to an email address which is monitored. If the Primary Server fails and the Secondary becomes active, an email notification of the event will be sent to the “`mail_from`” address.

15. If you have `acl_hosts` and `acl_host_enable` set on the server, you must

add the failover host to the list. Use the `qmgr` command:

Qmgr: s server acl_hosts+=<secondary server>

2.15.3 Failover Configuration for Windows

Under Windows, configure Server failover from the console of the hosts or through VNC.
Setting up the Server failover feature from a Remote Desktop environment will cause problems. In particular starting of the Server in either the primary host or secondary host would lead to the error:

error 1056: Service already running

even though `PBS_HOME\server_priv\server.lock` and `PBS_HOME\server_priv\server.lock.secondary` files are non-existent.

The following illustrates how PBS can be set up on Windows with the Server failover capability using the network share facility. That is, the primary and secondary Server/Scheduler will share a `PBS_HOME` directory that is located on a network share file system on a remote host. In this scenario a primary `pbs_server` is run on hostA, a secondary Server is run on hostB, and the shared `PBS_HOME` is set up on hostC using Windows network share facility.

Important: Note that hostC must be set up on a Windows Server-type platform.

1. Install PBS Windows on hostA and hostB accepting the default destination location of "`C:\Program Files\PBS Pro`".

2. Next stop all the PBS services on both hostA and hostB:

```
net stop pbs_server
net stop pbs_mom
net stop pbs_sched
net stop pbs_rshd
```

3. Now configure a shared PBS_HOME by doing the following:

a. Go to hostC; create a folder named e.g., C:\pbs_home.

b. Using Windows Explorer, right click select the C:\pbs_home file, and choose “Properties”.

c. Then select the "Sharing" tab, and click the check-button that says "Share this folder"; specify "Full Control" permissions for the local Administrators group on the local computer.

4. Next specify PBS_HOME for primary pbs_server on hostA and secondary Server on hostB by running the following on both hosts:

```
pbs-config-add
"PBS_HOME=\\hostC\pbs_home"
```

Now on hostA, copy the files from the local PBS home directory onto the shared PBS_HOME as follows:

xcopy /o /e “\Program Files\PBS Pro\home \\hostC\pbs_home”

5. Set up a local PBS_MOM_HOME by running the following command on both hosts:

pbs-config-add “PBS_MOM_HOME=C:\Program Files\PBS Pro\home”

6. Now create references to primary Server name and secondary Server name in the pbs.conf file by running on both hosts:

pbs-config-add “PBS_SERVER=hostA”
pbs-config-add “PBS_PRIMARY=hostA”

pbs-config-add “PBS_SECONDARY=hostB”

7. Now create references to primary Server name and secondary Server name in the `pbs.conf` file by running on execution and submission hosts:

   pbs-config-add “PBS_SERVER=hostA”
   pbs-config-add “PBS_PRIMARY=hostA”
   pbs-config-add “PBS_SECONDARY=hostB”

8. Set up the secondary Server so that it will only start the scheduler when it takes over from the Primary, and not when it is rebooted.

   On the secondary Server modify the `pbs.conf` file to start the scheduler by running:

   pbs-config-add “PBS_START_SCHED=1”

   Then go to the Control Panel->Administrative Tools->Services, and bring up the PBS_SCHED service dialog, select General tab, and specify "Manual" for Startup type.

   In this way, when the secondary host is rebooted, the scheduler won't automatically start up. Instead, the server can bring it up manually when it takes over for the primary Server. If the secondary server is told to take over and the primary host is still down, then the secondary server will start the scheduler via "net start pbs_sched".

9. Now start all the PBS services on hostA:

   net start pbs_mom
   net start pbs_server
   net start pbs_sched
   net start pbs_rshd

10. If you have `acl_hosts` and `acl_host_enable` set on the server, you must

add the failover host to the list. Use the `qmgr` command:

Qmgr: s server acl_hosts+=<secondary server>

11. Start the failover Server on hostB:

    net start pbs_server

    It's normal to get the following message:

    “PBS_SERVER could not be started”

    This is because the failover Server is inactive waiting for the primary Server to go down. If you need to specify a delay on how long the secondary Server will wait for the primary Server to be down before taking over, then you use `Start Menu->Control Panel->Administrative Tools->Services`, choosing `PBS_SERVER`, and specify under the “Start Parameters” entry box the value,

    “-F <delay_secs>”

    Then restart the secondary `pbs_server`. Keep in mind that the Services dialog does not remember the “Start Parameters” value for future restarts. The old default delay value will be in effect on the next restart.

12. Set the managers list on the primary Server so that when the secondary Server takes over, you can still do privileged tasks under the Administrator account or from a peer `pbs_server`:

    Qmgr: set server managers=“<*account that installed PBS*>@*,pbsadmin@*”

Important: Set up of the Server failover feature in Windows may encounter problems if performed from a Remote Desktop environment. In particular, starting the Server on either the primary host or secondary host would lead to the error:

error 1056 Service already running

even though `PBS_HOME\server_priv\server.lock` and `PBS_HOME\server_priv\server.lock.secondary` files are non-existent. To avoid this, configure Server failover from the console of the hosts or through VNC.

Important: Under certain conditions under Windows, the primary Server fails to take over from the secondary even after it is returned into the network. The workaround, should this occur, is to reboot the primary Server machine.

2.15.4 Failover: Normal Operation

The Primary Server and the Secondary Server may be started by hand, or via the system `init.d` script under UNIX, or using the Services facility under Windows. If you are starting the Secondary Server from the `init.d` script (UNIX only) and wish to change the failover delay, be sure to add the `-F` option to the `pbs_server`'s entry in the `init.d` script. Under Windows, specify -F as a start parameter given by the `Start-> Control Panel-> Administrator Tools-> Services-> PBS_SERVER` dialog.

The primary and the secondary server use different lock files:
primary: server.lock
secondary: server.lock.secondary.

It does not matter in which order the Primary and Secondary Servers are started.

Important: 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 (`server.lock`, `server.lock.secondary`) left in *PBS_HOME*/`server_priv` that need to be removed,

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.

When the Primary and Secondary Servers are initiated, the Secondary Server will periodically attempt to connect to the Primary. Once connected, it will send a request to register itself as the Secondary. The Secondary must register itself in order to take over should the Primary fail. The Primary will reply with information to allow the Secondary to use the license file location should it become active. The Primary and Secondary use the same license information, which is set through the Primary.

The Primary Server will then send "handshake" messages every few seconds to inform the Secondary Server that the Primary is alive. If the handshake messages are not received for the "take over" delay period, the Secondary will make one final attempt to reconnect to the Primary before becoming active. If the "take over" delay time is long, there may be a period, up to that amount of time, when clients cannot connect to either Server. If the delay is too short and there are transient network failures, then Secondary Server may attempt to take over while the Primary is still active.

While the Primary 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 if it is up.

If the Secondary Server becomes active, it will send email to the address specified in the Server attribute `mail_from`. The Secondary will inform the `pbs_mom` on the configured vnodes that it has taken over. The Secondary will attempt to connect to the Scheduler on the Primary host. If it is unable to do so, the Secondary will start a Scheduler on its host. The Secondary Server will then start responding to network connections and accepting requests from client commands such as `qstat` and `qsub`. If the secondary Server is started manually, it will not start its own scheduler. Since that is a manual operation, it is assumed that the manual operation will also start the Scheduler.

Job IDs will be identical regardless of which Server was running when the job was created, and will contain the name specified by `PBS_SERVER` in `pbs.conf`.

In addition to the email sent when a Secondary Server becomes active, there is one other method to determine which Server is running. The output of a "`qstat -Bf`" command includes the "`server_host`" attribute whose value is the name of the host on which the Server is running.

When a user issues a PBS command directed to a Server that matches the name given by `PBS_SERVER`, the command will normally attempt to connect to the Primary Server. If it is unable to connect to the Primary Server, the command will attempt to connect to the Secondary Server (if one is configured). If this connection is successful, then the command will create a file referencing the user executing the command. (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 login name.) Any future command execution will detect the presence of that file and attempt to connect to the Secondary Server first. This eliminates the delay in attempting to connect to the down Server. If the command cannot connect to the Secondary Server, and can connect to the Primary, the command will remove the above referenced file.

2.15.5 Failover: Manual Shutdown

Any time the Primary Server exits, because of a fault, or because it was told to shut down by a signal or the `qterm` command, the Secondary Server will become active.

If you wish to shut down the Primary Server and not have the Secondary Server become active, you must either:

1 Use the `-f` option on the `qterm` command. This causes the Secondary Server to exit as well as the Primary; or

2 Use the `-i` option on the `qterm` command, this causes the Secondary Server to remain running but inactive (standby state); or

3 Manually kill the Secondary Server before terminating the Primary Server (via sending any of `SIGKILL`, `SIGTERM`, or `SIGINT`).

If the Primary Server exits causing the Secondary Server to become active and you then restart the Primary Server, it will notify the Secondary Server to restart and become inactive. You need not terminate the active Secondary Server before restarting the Primary. However, be aware that if the Primary cannot contact the Secondary due to network outage, it will assume the Secondary is *not* running. The Secondary will remain active resulting in two active Servers. If you need to shut down and restart the Secondary Server while it is active, and wish to keep it active, then use the `pbs_server` with the `-F` option and a delay value of “-1”:

```
pbs_server -F -1
```

The negative one value directs the Secondary Server to become active immediately. It will still make one attempt to connect to the Primary Server in case the Primary is actually up. The default delay is 30 seconds.

2.15.6 Failover and Route Queues

When setting up a Server route queue whose destination is in a failover configuration, it is necessary to define a second destination that specifies the same queue on the Secondary Server.

For example, if you already have a routing queue created with a destination as shown:

```
Qmgr: set queue r66
route_destinations=workq@primary.xyz.com
```

you need to add the following additional destination, naming the secondary Server host:

```
Qmgr: set queue r66
route_destinations+=workq@secondary.xyz.com
```

2.15.7 Failover and Peer Scheduling

If the Server being configured is also participating in Peer Scheduling, both the Primary and Secondary Servers need to be identified as peers to the Scheduler. For details, see section 4.18.3 “Peer Scheduling and Failover Configuration” on page 230.

2.16 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
```

2.17 Server Support for Globus

If Globus support is enabled, then an entry must be manually entered into the PBS nodes file (*PBS_HOME*/`server_priv/nodes`) with `:gl` appended to the name. This is the only case in which two vnodes may be defined with the same vnode name. One may be a Globus vnode (MOM), and the other a non-Globus vnode. If you run both a Globus MOM and a normal MOM on the same site, the normal PBS MOM must be listed first in your nodes file. If not, some scheduling anomalies could appear.

Important: Globus support is not currently available on Windows.

2.18 Configuring the Server for FLEX Licensing

The PBS server must be configured for FLEX licensing. You must set the location where PBS will look for the license server, by setting the server attribute pbs_license_file_location. The other server licensing attributes have defaults, but you may wish to set them as well. See section 5.4 “Configuring PBS for Licensing” on page 91 in the PBS Professional Installation & Upgrade Guide.

You may also wish to have redundant license servers. See the Altair License Management System 9.0 Installation Guide.

Chapter 3

Configuring MOM

The installation process creates a basic MOM configuration file which contains the minimum necessary in order to run PBS jobs. This chapter describes the MOM configuration files, and explains all the options available to customize the PBS installation to your site.

The organization of this chapter has changed. Information specific to configuring machines such as the Altix is presented in section 3.9 “Configuring MOM for Machines with cpusets” on page 144.

3.1 Introduction

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.

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 manages one or more 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. See section 2.8 “Vnodes: Virtual Nodes” on page 48.

The MOM's error log file is in PBS_HOME/mom_logs. The MOM writes an error message in its log file when it encounters any error. 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.

For information on starting and stopping MOM, see section 6.5.3 “Manually Starting MOM” on page 278.

3.1.1 Single- vs. Multi-vnoded Systems

The following section contains information that applies to all PBS MOMs. The PBS MOM pbs_mom.cpuset has extensions to take manage multi-vnoded systems such as the Altix. These systems can be subdivided into more than one virtual node, or vnode. PBS manages each vnode as if it were a host. While the information in this section is true for all MOMs, any information that is specific to multi-vnoded systems is in section 3.9 “Configuring MOM for Machines with cpusets” on page 144.

3.1.2 Hyperthreaded Systems

On Linux machines that have hyperthreading, PBS will report the number of CPUs that the operating system reports. If you do not wish to use hyperthreading, you can configure PBS to use the number of physical CPUs. This is done by setting resources_available.ncpus=<number of physical CPUs>.

3.2 MOM Configuration Files

The behavior of each MOM is controlled through its 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.

MOM's configuration information can be contained in configuration files of three types: default, PBS reserved, and site-defined. The default configuration file is usually PBS_HOME/mom_priv/config. PBS reserved configuration files are created by PBS and are prefixed with "PBS". Site-defined configuration files are those created by the site administrator.

Any PBS reserved MOM configuration files are only created when PBS is started, not when the MOM is started. Therefore, 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 in order to re-create the PBS reserved MOM configuration files.

When MOM is started, it will open its default 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 section 6.5.3 “Manually Starting MOM” on page 278.

The files are processed in this order:

    The default configuration file
    PBS reserved configuration files
    Site-defined 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.

3.2.1 Creation of Site-defined MOM Configuration Files

To change the cpuset flags, create a file "update_flags" containing only

```
cpuset_create_flags CPUSET_CPU_EXCLUSIVE
```

then use the `pbs_mom -s insert <script> <filename>` option to create the script:

```
pbs_mom -s insert update_script update_flags
```

The script `update_script` is the new site-defined configuration file. Its contents will override previously-read `cpuset_create_flags` settings.

Configuration files can be listed, added, deleted and displayed using the `-s` option. An attempt to create or remove a file with the "PBS" prefix will result in an error. See section 6.5.3 “Manually Starting MOM” on page 278 for information about pbs_mom options.

MOM's configuration files can use the syntax shown below in section 3.2.2 “Syntax and Contents of Default Configuration File” on page 111, or the syntax for describing vnodes shown in section 3.9.3.2 “Syntax of Version 2 PBS Reserved Configuration Files” on page 145.

3.2.1.1 Location of MOM’s Configuration Files

The default configuration file is in PBS_HOME/mom_priv/.

PBS places PBS reserved and site-defined configuration files in an area that is private to each installed instance of PBS. This area may change with future releases. Do not attempt to manipulate these files directly. This area is relative to the default PBS_HOME. Note that the `-d` option changes where MOM looks for PBS_HOME, and using this option will prevent MOM from finding any but the default configuration file. If you use the -d option, MOM will look in the wrong place for any PBS reserved and site-defined files.

Do not directly create PBS reserved or site-defined configuration files; instead, use the `pbs_mom -s` option. See section 6.5.3 “Manually Starting MOM” on page 278 for information on `pbs_mom`.

The `-c` option will change which default configuration file MOM reads.

Site-defined configuration files can be moved from one installed instance of PBS to another. Do not move PBS reserved configuration files. To move a set of site-defined 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
site-defined files.

2.Use the -s show directive with each site-defined 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 site-defined configuration file.

3.2.2 Syntax and Contents of Default Configuration File

Configuration files with this syntax list local resources and initialization values for MOM. Local resources are either static, listed by name and value, or externally-provided, listed by name and command path. Local static resources are for use only by the scheduler. They do not appear in a `pbsnodes -a` query. See the -c option. Do not change the syntax of the default configuration file.

Each configuration item is listed on a single line, with its parts separated by white space. Comments begin with a hashmark ("#").

The default 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.

3.2.2.1 Externally-provided Resources

Externally-provided resources use a shell escape to run a command. These resources are described with a name and value, where the first character of the value is an exclamation mark ("!"). The remainder of the value is the path and command to execute.

Parameters in the command beginning with a percent sign ("%") can be replaced when the command is executed. For example, this line in a configuration file describes a resource named "escape":

escape !echo %xxx %yyy

If a query for the "escape" resource is sent with no parameter replacements, the command executed is "echo %xxx %yyy". If one parameter replacement is sent, "escape[xxx=hi there]", the command executed is "echo hi there %yyy". If two parameter replacements are sent, "escape[xxx=hi][yyy=there]", the command executed is "echo hi there". If a parameter replacement is sent with no matching token in the command line, "escape[zzz=snafu]", an error is reported.

3.2.2.2 Initialization Values

Initialization value directives have names beginning with a dollar sign ("$"). They are listed here:

$action <default_action> <timeout> <new_action>

Replaces the `default_action` for an event with the site-specified `new_action`. `timeout` is the time allowed for `new_action` to run. This is the complete list of values for `default_action`:

Table 1: How $action is Used

default_action	Result
checkpoint	Run `new_action` in place of the periodic job checkpoint, after which the job continues to run.
checkpoint_abort	Run `new_action` to checkpoint the job, after which the job is terminated.
multinodebusy	Used with cycle harvesting and multi-vnode jobs. Changes default behavior when a vnode becomes busy. Instead of allowing the job to run, the job is requeued. The `new_action` is requeue.
restart	Runs `new_action` in place of restart.
terminate	Runs `new_action` in place of SIGTERM or SIGKILL when MOM terminates a job.

`$checkpoint_path <path>`

MOM will write checkpoint files in the directory given by `path`. This path can be absolute or relative to `PBS_HOME/mom_priv`.

`$clienthost <hostname>`

`hostname` is added to the list of hosts which will be allowed to connect to MOM as long as they are using a privileged port. For example, this will allow the hosts "fred" and "wilma" to connect to MOM:

```
$clienthost fred
$clienthost wilma
```

The following hostnames are added to $clienthost automatically. The server and the localhost are automatically added to $clienthost. If configured, the secondary server is also added to $clienthost. The server sends each MOM a list of the hosts in the nodes file, and these are added internally to $clienthost. None of these hostnames need to be listed in the configuration file.

The hosts in the nodes file make up a "sisterhood" of machines. Any one of the sisterhood will accept connections from within the sisterhood. The sisterhood must all use the same port number.

`$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 this factor to a decimal value greater than 1.0. For example:

```
$cputmult 1.5
```

If MOM's system is slower, set this factor to a value between 1.0 and 0.0. For example:

```
$cputmult 0.75
```

`$dce_refresh_delta <delta>`

Defines the number of seconds between successive refreshings of a job's DCE login context. For example:

$dce_refresh_delta 18000

`$enforce <limit>`

MOM will enforce the given limit. Some limits have associated values, and appear in the configuration file like this:

$enforce variable_name value

See section 3.8 “Resource Limit Enforcement” on page 137.

`$enforce mem` MOM will enforce each job's memory limit. See section 3.8 “Resource Limit Enforcement” on page 137.

`$enforce cpuaverage`

MOM will enforce `ncpus` when the average CPU usage over a job's lifetime usage is greater than the job's limit. See section 3.8.2.1 “Average CPU Usage Enforcement” on page 141.

`$enforce average_trialperiod <seconds>`

Modifies cpuaverage. Minimum number of seconds of job walltime before enforcement begins. Default: 120. Integer.
See section 3.8.2.1 “Average CPU Usage Enforcement” on page 141.

`$enforce average_percent_over <percentage>`

Modifies cpuaverage. Gives percentage by which a job may exceed its ncpus limit. Default: 50. Integer.
See section 3.8.2.1 “Average CPU Usage Enforcement” on page 141.

`$enforce average_cpufactor <factor>`

Modifies cpuaverage. The ncpus limit is multiplied by factor to produce actual limit. Default: 1.025.

Float. See section 3.8.2.1 “Average CPU Usage Enforcement” on page 141.

`$enforce cpuburst`

MOM will enforce the ncpus limit when CPU burst usage exceeds the job's limit. See section 3.8.2.2 “CPU Burst Usage Enforcement” on page 141.

`$enforce delta_percent_over <percentage>`

Modifies cpuburst. Gives percentage over limit to be allowed. Default: 50. Integer. See section 3.8.2.2 “CPU Burst Usage Enforcement” on page 141.

`$enforce delta_cpufactor <factor>`

Modifies cpuburst. The ncpus limit is multiplied by factor to produce actual limit. Default: 1.5. Float. See section 3.8.2.2 “CPU Burst Usage Enforcement” on page 141.

`$enforce delta_weightup <factor>`

Modifies cpuburst. Weighting factor for smoothing burst usage when average is increasing. Default: 0.4. Float. See section 3.8.2.2 “CPU Burst Usage Enforcement” on page 141.

`$enforce delta_weightdown <factor>`

Modifies cpuburst. Weighting factor for smoothing burst usage when average is decreasing. Default: 0.4. Float. See section 3.8.2.2 “CPU Burst Usage Enforcement” on page 141.

`$ideal_load <load>`

Defines the load below which the host is not considered to be busy. Used with the $max_load directive. No default. Float. Example:

$ideal_load 1.8

Use of $ideal_load adds a static resource called "ideal_load", which is only internally visible.

`$jobdir_root <directory>`

Directory under which PBS creates job-specific staging and execution directories. PBS creates a job's staging directory when the job's `sandbox` attribute is set to PRIVATE. Defaults to job owner'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. See section 6.14 "The Job's Staging and Execution Directories" on page 330.

`$kbd_idle <idle_wait> <min_use> <poll_interval>`

Declares that the host will be used for batch jobs during periods when the keyboard and mouse are not in use.

The host must be idle for a minimum of idle_wait seconds before being considered available for batch jobs. No default. Integer.

The host must be in use for a minimum of min_use seconds before it becomes unavailable for batch jobs. Default: 10. Integer.

Mom checks for activity every poll_interval seconds. Default: 1. Integer.

Example:

$kbd_idle 1800 10 5

`$logevent <mask>`

Sets the mask that determines which event types are logged by pbs_mom. To include all debug events, use 0xffffffff.

Table 2: Log Events

Name	Hex Val	Message Category
PBSE_ERROR	0001	Internal errors
PBSE_SYSTEM	0002	System errors
PBSE_ADMIN	0004	Administrative events
PBSE_JOB	0008	Job-related events
PBSE_JOB_USAGE	0010	Job accounting info
PBSE_SECURITY	0020	Security violations
PBSE_SCHED	0040	Scheduler events
PBSE_DEBUG	0080	Common debug messages
PBSE_DEBUG2	0100	Uncommon debug messages
PBSE_RESV	0200	Reservation-related info
PBSE_DEBUG3	0400	Rare debug messages

`$max_check_poll <seconds>`

Maximum time between polling cycles, in seconds. Must be greater than zero. Integer.

`$min_check_poll <seconds>`

Minimum time between polling cycles, in seconds. Must be greater than zero and less than $max_check_poll. Integer.

`$max_load <load> [suspend]`

Defines the load above which the host is considered to be busy. Used with the `$ideal_load` directive. No default. Float. Example:

$max_load 3.5 suspend

Use of `$max_load` adds a static resource to the vnode called "max_load", which is only internally visible.

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.

`$prologalarm <timeout>`

Defines the maximum number of seconds the prologue and epilogue may run before timing out. Default: 30. Integer. Example:

$prologalarm 30

`$restart_background <true|false>`

Controls how MOM runs a restart script after checkpointing a job.
When this option is set to true, MOM forks a child which runs the restart script. The child returns when all restarts for all the local tasks of the job are done. MOM does not block on the restart. When this option is set to false, MOM runs the restart script and waits for the result. Boolean. Default: false.

`$restart_transmogrify <true|false>`

Controls how MOM runs a restart script after checkpointing a job. When this option is set to true, MOM runs the restart script, replacing the session ID of the original task's top process with the session ID of the script.

When this option is set to false, MOM runs the restart script and waits for the result. The restart script must restore the original session ID for all the processes of each task so that MOM can continue to track the job.

When this option is set to false and the restart uses an external command, the configuration parameter restart_background
is ignored and treated as if it were set to true, preventing MOM from blocking on the restart.

Boolean. Default: false

`$restrict_user <value>`
Controls whether users not submitting jobs have access to this machine. If value is "on", restrictions are applied. The interval between when PBS applies restrictions can be at most 10 seconds. See `$restrict_user_exceptions` and `$restrict_user_maxsysid`. Boolean. Default: off.

`$restrict_user_exceptions <user_list>`
Comma-separated list of users who are exempt from access restrictions applied by `$restrict_user`. Leading spaces within each entry are allowed. Maximum number of names in list is 10.

`$restrict_user_maxsysid <value>`
Any user with a numeric user ID less than or equal to value is exempt from restrictions applied by `$restrict_user`.

If `$restrict_user` is on and no value exists for `$restrict_user_maxsysid`, PBS looks in `/etc/login.defs for` SYSTEM_UID_MAX for the value. If there is no maximum ID, it looks for SYSTEM_MIN_UID, and uses that value minus 1. Otherwise the default is used.

Integer. Default: 999.

`$restricted <hostname>`

The hostname is added to the list of hosts which will be allowed to connect to MOM from a non-privileged port. Hostnames can be wildcarded. For example, to allow queries from any host from the domain "xyz.com":

$restricted   *.xyz.com

Queries from the hosts in the $restricted list are only allowed access to information internal to the host managed by this MOM, such as load average, memory available, etc. They may not run shell commands. No machines are added automatically to this list.

`$suspendsig <suspend_signal> [resume_signal]`

Alternate signal suspend_signal is used to suspend jobs instead of SIGSTOP. Optional resume_signal is used to resume jobs instead of SIGCONT.

`$tmpdir <directory>`

Location where each job's scratch directory will be created. The TMPDIR environment variable contains the path to the scratch directory. Default: /tmp. For example:

$tmpdir /memfs

`$usecp <hostname:source_prefix> <destination_prefix>`

MOM will use /bin/cp (or xcopy on Windows) to stage in/out files or deliver output when the source and destination are both on the local host. In addition, the administrator can use $usecp to list other source locations that can be directly copied to/from local destinations. Both source_prefix and destination_prefix are absolute pathnames of directories, not files. For example:

$usecp*.example.com:/home/ /home/

This says any file available remotely under the /home/ directory is also directly available to the MOM in the /home/ directory.

$usecpHostA:/users/work/myproj/ \
/sharedwork/proj_results/

This says that a staging or output reference to a file under
/users/work/myproj/ on HostA should instead refer to a file under /sharedwork/proj_results/ on the MOM.

`$wallmult <factor>`

Each job's walltime usage is multiplied by this factor. For example:

$wallmult 1.5

3.2.2.3 Static MOM Resources

Local static resources are for use only by the scheduler. They do not appear in a `pbsnodes -a` query. Static resources local to the MOM are described one resource to a line, with a name and value separated by white space. For example, tape drives of different types could be specified by:

tape3480 4

tape3420 2

tapedat 1

tape8mm 1

`memreserved <megabytes>`

The amount of per-vnode memory reserved for system overhead. This much memory is deducted from the value of resources_available.mem for each vnode managed by this MOM. Default is 0MB. For example,

memreserved 16

3.2.2.4 Windows Notes

If the argument to a MOM option is a pathname containing a space, enclose it in double quotes as in the following:

hostn !"\Program Files\PBS Pro\exec\bin\hostn" *host*

3.3 Configuring MOM's Polling Cycle

MOM's polling cycle is set by `$min_check_poll` and `$max_check_poll`. The interval between each poll starts at `$min_check_poll` and increases with each cycle until it reaches `$max_check_poll`, after which it remains the same. The amount by which the cycle increases is 1/20 of the difference between `$max_check_poll` and `$min_check_poll`.

MOM polls for resource usage for cput, walltime, mem and ncpus. See section 3.8 "Resource Limit Enforcement" on page 137. Job-wide limits are enforced by MOM using polling. See section 3.8.1 "Job Memory Limit Enforcement on UNIX" on page 138. MOM can enforce cpuaverage and cpuburst resource usage; see section 3.8.2.1 "Average CPU Usage Enforcement" on page 141 and section 3.8.2.2 "CPU Burst Usage Enforcement" on page 141.

MOM enforces the $restrict_user access restrictions on a polling cycle which can be set to a maximum of 10 seconds. See section 3.7 "Restricting User Access to Execution Hosts" on page 135.

Cycle harvesting has its own polling interval. See the information for $kbd_idle in section 3.2.2.2 "Initialization Values" on page 112.

3.4 Configuring MOM Resources

3.4.1 Static MOM Resources

Configure static vnode-level resources using `qmgr`.
Example:

```
Qmgr: set node VNODE resources_available.RES
= <value>
```

While it is possible to configure static resources in the MOM configuration file, it is not recommended. Qmgr is preferred because (1) the change takes effect immediately, as opposed to having to send a HUP signal to MOM; and (2) all such static resources can be centrally managed and viewed via `qmgr`. For more information on creating site-specific resources, see Chapter 9, “Customizing PBS Resources” on page 237.

That being said, to specify static resource names and values in the MOM configuration file, you can add a list of resource name/value pairs, one pair per line, separated by white space.

3.4.2 Dynamic MOM Resources

Configure dynamic vnode-level resources by adding shell escapes to the MOM configuration file, *PBS_HOME*/`mom_priv/config` . The primary use of this feature is to add site-specific resources, such as software application licenses. The form is:

```
RESOURCE_NAME !path-to-command
```

The `RESOURCE_NAME` specified should be the same as the corresponding entry in the Server’s *PBS_HOME*/`server_priv/resourcedef` file. See Chapter 9, “Customizing PBS Resources” on page 237 and section 5.7 “Application Licenses” on page 256.

3.5 Configuring MOM for Site-Specific Actions

3.5.1 Site-specific Job Termination Action

The default behavior of PBS is for MOM to terminate a job when the job's usage of a resource exceeds the limit requested or when the job is deleted by the Server on shutdown or because of a `qdel` command. However, a site may specify a script (or program) to be run by `pbs_mom` in place of the normal `SIGTERM/SIGKILL` action when MOM is terminating a job under the above conditions. This action takes place on terminate from exceeding resource limits or from usage of the `qdel` command. The script is defined by adding the following parameter to MOM's config file:

`$action terminate` *TIME_OUT* !*SCRIPT_PATH [ARGS]*

Where *`TIME_OUT`* is the time, in seconds, allowed for the script to complete.

`SCRIPT_PATH` is the path to the script. If it is a relative path, it is evaluated relative to the `PBS_HOME/mom_priv` directory.

Important: Under Windows, *`SCRIPT_PATH`* must have a "`.bat`" suffix since it will be executed under the Windows command prompt `cmd.exe`. If the *`SCRIPT_PATH`* 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.

`ARGS` are optional arguments to the script. Values for *`ARGS`* may be: any string not starting with '%'; or `%keyword`, which is replaced by MOM with the corresponding value:

%jobid	job id
%sid	session id of task (job)
%uid	execution uid of job
%gid	execution gid of job
%login	login name associated with uid
%owner	job owner "name@host"
%auxid	aux id (system dependent content)

If the script exits with a zero exit status (before the time-out period), PBS will 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. If the script exits with a non-zero exit status, the job will be sent `SIGKILL` by PBS. If the script does not complete in the time-out period, it is aborted and the job is sent `SIGKILL`. A *`TIME_OUT`* value of 0 is an infinite time-out.

A UNIX example:

```
$action terminate 60 !endjob.sh %sid %uid
%jobid
```

or

```
$action terminate  0 !/bin/kill -13 %sid
```

A similar Windows example:

```
$action terminate 60 !endjob.bat %sid %uid
%jobid
```

or

```
$action terminate  0 !"C:/Program Files/PBS
Pro/exec/bin/pbskill" %sid
```

The first line in both examples above sets a 60 second timeout value, and specifies that `PBS_HOME/mom_priv/endjob.sh` (`endjob.bat` under Windows) should be executed with the arguments of the job's session ID, user ID, and PBS jobs ID. The third line in the first (UNIX) example simply calls the system `kill` command with a specific signal (13) and the session ID of the job. The third line of the Windows example calls the PBS-provided `pbskill` command to terminate a specific job, as specified by the session id (`%sid`) indicated.

3.5.2 Site-Specific Job Checkpoint and Restart

The PBS Professional site-specific job checkpoint facility allows an Administrator to replace the built-in checkpoint facilities of PBS Professional with a site-defined external command. This is most useful on computer systems that do not have OS-level checkpointing. This feature is used by setting these MOM configuration parameters.

$action checkpoint *TIME_OUT* !*SCRIPT_PATH ARGS* [...]
$action checkpoint_abort *TIME_OUT* !*SCRIPT_PATH ARGS* [...]
$action restart *TIME_OUT* !*SCRIPT_PATH* [*ARGS* ...]

The `checkpoint` parameter specifies that the script in `SCRIPT_PATH` is run, and the job is left running. This script is called once for each of the job's tasks, and is supplied by the site. The script must take care of everything necessary to checkpoint the job and restart it.

The `checkpoint_abort` parameter specifies that the script in `SCRIPT_PATH` is run, but the job is terminated. This script is called once for each of the job's tasks, and is supplied by the site. The script must handle everything necessary to checkpoint the job and restart it.

The `restart` parameter specifies the script to be used to restart the job. This script is called once for each of the job's tasks, and is supplied by the site. When the job is restarted, it will be running on the same machine as before, with the same priority.

TIME_OUT is the time (in seconds) allowed for the script (or program) to complete. If the script does not complete in this period, it is aborted and handled in the same way as if it returned a failure. This does not apply if `restart_transmogrify` is "true" (see below), in which case, no time check is performed.

SCRIPT_PATH is the path to the script. If it is a relative path, it is evaluated relative to the `PBS_HOME/mom_priv` directory.

ARGS are the arguments to pass to the script. The following *ARGS* are expanded by PBS:

`%globid`	Global ID
`%jobid`	Job ID
`%sid`	Session ID
`%taskid`	Task ID
`%path`	File or directory name to contain checkpoint files

PBS uses the following MOM configuration parameters to control how restart scripts are run. See "$restart_background <true|false>" on page 118 and "$restart_transmogrify <true|false>" on page 118.

$restart_background (true|false)
$restart_transmogrify (true|false)

The MOM configuration parameter `restart_background` is a boolean flag that modifies how MOM performs a restart. When the flag is "false" (the default), MOM runs the restart operation and waits for the result. When the flag is "true", restart operations are done by a child of MOM which only returns when all the restarts for all the local tasks of a job are done. The parent (main) MOM can then continue processing without being blocked by the restart.

The MOM configuration parameter `restart_transmogrify` is a boolean flag that controls how MOM launches the restart script/program. When the flag is “false” (the default) MOM will run the restart script and block until the restart operation is complete (and return success or appropriate failure). In this case the restart action must restore the original session ID for all the processes of each task or MOM will no longer be able to track the job. Furthermore, if `restart_transmogrify` is “false” and restart is being done with an external command, the configuration parameter `restart_background` will be ignored and the restart will be done as if the setting of `restart_background` was “true”. This is to prevent a script that hangs from causing MOM to block. If `restart_transmogrify` is “true”, MOM will run the restart script/program in such a way that the script will “become” the task it is restarting. In this case the restart action script will replace the original task's top process. MOM will replace the session ID for the task with the session ID from this new process. If a task is checkpointed, restarted and checkpointed again when `restart_transmogrify` is “true”, the session ID passed to the second checkpoint action will be from the new session ID.

3.5.3 Guidelines for Creating Local Checkpoint Action

This section provides a set of guidelines the Administrator should follow when creating a site-specific job checkpoint / restart program (or script). PBS will initiate the checkpoint program/script for each running task of a job. This includes all the vnodes where the job is running. The following environment variables will be set:

Table 3:

GID	HOME	LOGNAME	PBS_GLOBID
PBS_JOB\ COOKIE	PBS_JOBID	PBS_JOBNA ME	PBS_MOMPO RT
PBS_NODEFILE	PBS_NODE\ NUM	PBS_QUEUE	PBS_SID
PBS_TASKNUM	SHELL	UID	USER

The checkpoint command should expect and handle the following inputs:

Global ID
Job ID

Session ID
Task ID
File or directory name to contain checkpoint files

The restart command should return success or failure error codes, and expect and handle as input a file/directory name. The restart script has access to the PBS_NODEFILE environment variable.

Both the checkpoint and restart scripts/programs should block until the checkpoint/restart operation is complete. When the script completes, it should indicate success or failure by returning an appropriate exit code and message. To PBS, an exit value of 0 indicates success, and a non-zero return indicates failure.

Note that when the MOM configuration parameter `restart_transmogrify` is set to "false" the restart action must restore the original session ID for all the processes of each task or MOM will no longer be able to track the job. If the parameter `restart_transmogrify` is set to "true", when the restart script for a task exits, the task will be considered done, and the restart action *`TIME_OUT`* will not be used.

Note: checkpointing is not supported for job arrays. On systems that support checkpointing, subjobs are not checkpointed; instead they run to completion.

3.6 Configuring Idle Workstation Cycle Harvesting

"Harvesting" of idle workstation cycles is a method of expanding the available computing resources of your site by automatically including in your complex unused workstations that otherwise would have sat idle. This is particularly useful for sites that have a significant number of workstations that sit on researchers' desks and are unused during the nights and weekends. With this feature, when the "owner" of the workstation isn't using it, the machine can be configured to be used to run PBS jobs. Detection of "usage" can be configured to be based upon system load average or by keystroke activity (as discussed in the following two sections below). Furthermore, cycle harvesting can be configured for all jobs, single-vnode jobs only, and/or with special treatment for multi-vnode (parallel) jobs. See section 3.6.4 "Restrictions on Cycle Harvesting" on page 134 for details.

3.6.1 Cycle Harvesting Based on Load Average

Cycle harvesting based on load average is load balancing based on load average. You set each workstation's max_load and ideal_load. When max_load is exceeded, the node is marked as "state=busy". It will show up this way in pbsnodes and the scheduler will not place jobs on busy nodes. When the load drops below ideal_load, the state changes back to "free".

To set up cycle harvesting of idle workstations based on load average, perform the following steps:

Step 1 If PBS is not already installed on the target execution workstations, do so now, selecting the execution-only install option. (See Chapter 4 of this manual for details.)

Step 2 Edit the `PBS_HOME/mom_priv/config` configuration file on each target execution workstation, adding the two load-specific configuration parameters with values appropriate to your site.

$max_load 5
$ideal_load 3

Then HUP the MOM:

kill -HUP <pbs_mom PID>

Step 3 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

If you wish to oversubscribe the vnode's CPU(s), set its `resources_available.ncpus` to a higher number.

Then HUP the scheduler:

kill -HUP <pbs_sched PID>

Step 4 Set the vnode's `resv_enable` attribute to False, to prevent the vnode from being used for reservations.

Step 5 Set the vnode's `no_multinode_jobs` attribute to False, to prevent the vnode from stalling multi-chunk jobs.

3.6.2 Cycle Harvesting Based on Keyboard/Mouse Activity

If a system is configured for keyboard/mouse-based cycle harvesting, it becomes available for batch usage by PBS if its keyboard and mouse remain unused or idle for a certain period of time. The workstation will be shown in state "free" when the status of the vnode is queried. If the keyboard or mouse is used, the workstation becomes unavailable for batch work and PBS will suspend any running jobs on that workstation and not attempt to schedule any additional work on that workstation. The workstation will be shown in state "busy", and any suspended jobs will be shown in state "U".

Important: Jobs on workstations that become *busy* will not be migrated; they will remain on the workstation until they complete execution, are rerun, or are deleted.

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 illustrates the method and support per system.

Table 4:

System	Status	Method	Reference
AIX	supported	pbs_idled	See section 3.6.3.
FreeBSD	unsupported	pbs_idled	See section 3.6.3.

Table 4:

System	Status	Method	Reference
HP-UX 11	supported	device	See below
Linux	supported	device	See below
Solaris	supported	device	See below
Windows XP Pro	supported	other	See below
Windows 2003 Server	supported	other	See below

The cycle harvesting feature is enabled via a single entry in `pbs_mom`'s `config` file, `$kbd_idle`, and takes up to three parameters, as shown below.

$kbd_idle *idle_wait* [*min_use* [*poll_interval*]]

These three parameters, representing time specified in seconds, control the transitions between *free* and *busy* states. Definitions follow.

`idle_wait` time (in seconds) that the workstation keyboard and mouse must be idle before the workstation becomes available to PBS.

`min_use` time period during which the keyboard or mouse must remain busy before the workstation "stays" unavailable. This is used to keep a single key stroke or mouse movement from keeping the workstation busy.

`poll_interval` frequency of checking the state of the keyboard and mouse.

After changing each MOM's configuration file, HUP the MOM:

```
kill -HUP <pbs_mom PID>
```

Let us consider the following example.

$kbd_idle 1800 10 5

Adding the above line to MOM's `config` file directs PBS to mark the workstation as *free* if the keyboard and mouse are idle for 30 minutes (1800 seconds), to mark the workstation as *busy* if the keyboard or mouse are used for 10 consecutive seconds, and the state of the keyboard/mouse is to be checked every 5 seconds.

The default value of *min_use* is 10 seconds, the default for *poll_interval* is 1 second. There is no default for *idle_wait*; setting it to non-zero is required to activate the cycle harvesting feature.

Elaborating on the above example will help clarify the role of the various times. Let's start with a workstation that has been in use for some time by its owner. The workstation is shown in state *busy*. Now the owner goes to lunch. After 1800 seconds (30 minutes), the system will change state to *free* and PBS may start assigning jobs to run on the system. At some point after the workstation has become *free* and a job is started on it, 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 shown being in state "U" and the workstation is marked *busy*. If, after 10 seconds have passed and there is no additional keyboard/mouse activity, the job is resumed and the workstation again is shown as either *free* (if any CPUs are available) or *job-busy* (if all CPUs are in use.) However, if keyboard/mouse activity continued during that 10 seconds, then the workstation would remain *busy* and the job would remain suspended for at least the next 1800 seconds.

3.6.3 Cycle Harvesting on Machines with X-Windows

On some systems cycle harvesting is simple to implement as the console, keyboard, and mouse device access times are updated by the operating system periodically. The PBS MOM process takes note of that and marks the vnode busy if any of the input devices are in use. On other systems, however, this data is not available. (See table in section 3.6.2 above.) In such cases, PBS must monitor the X-Window System in order to obtain interactive idle time. To support this, there is a PBS X-Windows monitoring process called `pbs_idled`. This program runs in the background and monitors X and reports to the `pbs_mom` whether the vnode is idle or not.

Because of X-Windows security, running `pbs_idled` requires more modification than just installing PBS. First, a directory must be made for `pbs_idled`. This directory must have the same permissions as `/tmp` (i.e. mode 1777). This will allow the `pbs_idled` to create and update files as the user, which is necessary because the program will be running as the user. For example:

```
on Linux:
mkdir /var/spool/PBS/spool/idledir
chmod 1777 /var/spool/PBS/spool/idledir

on UNIX:
mkdir /usr/spool/PBS/spool/idledir
chmod 1777 /usr/spool/PBS/spool/idledir
```

Next, turn on keyboard idle detection in the MOM `config` file:

```
$kbd_idle 300
```

Lastly, `pbs_idled` needs to be started as part of the X-Windows startup sequence. The best and most secure method of installing `pbs_idled` is to insert it into 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. The startup line for `pbs_idled` must be before that of the window manager. It is also very important that `pbs_idled` is run in the background. On systems that use `Xsession` to start desktop sessions, a line invoking `pbs_idled` should be inserted near the top of the file. `pbs_idled` is located in `$PBS_EXEC/sbin`. For example, the following line should be inserted in a Linux `Xsession` file:

```
/usr/pbs/sbin/pbs_idled &
```

Note that if access to the system-wide `Xsession` file is not available, `pbs_idled` may be added to every user's personal `.xsession, .xinitrc, or .sgisession` file (depending on the local OS requirements for starting X-windows programs upon login).

3.6.4 Restrictions on Cycle Harvesting

Cycle harvesting is incompatible with some kinds of jobs, including multi-host jobs and jobs in reservations. If one of the hosts being used for a parallel job running on several hosts is being used for cycle harvesting at a time when the user types at the keyboard, it will delay job execution for the entire job because the tasks running on that host will be suspended. 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 job(s) in the reservation, defeating the purpose of making a reservation.

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 request more than one chunk.

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.

3.6.5 Parallel Jobs With Cycle Harvesting

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 the inter-process communications. Thus the default action for a job which uses multiple hosts when one or more of the hosts becomes busy, is to leave the job running.

It is possible, however, to specify that the job should be requeued (and subsequently re-scheduled to run elsewhere) when any of the hosts (vnodes) on which the job is running becomes *busy.* To enable this action, the Administrator must 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.

Important: Jobs which are not rerunnable (i.e. those submitted with the `qsub -rn` option) will be killed if the requeue action is configured and a vnode becomes busy.

3.6.6 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 will be suspended as well.

However, if the job has PBS file staging parameters (i.e. `stageout=file1`...), the file transfer will not be suspended. This is because the file staging occurs as part of the post-execution (or "`Exiting`" state, after the `epilogue` is run), and is not subject to suspension. (For more information on PBS file staging, see the **PBS Professional User's Guide**.)

3.6.7 Cycle Harvesting on Windows

Under Windows, when a machine becomes "busy" because the keyboard is being used, the effect on the job is different. Instead of being suspended, the job has its priority lowered from Normal to Low. For example, you submit a job and it begins to run on a workstation, and the CPU loading on that machine goes to 100%. Then you move the mouse: you'll see that the CPU loading is still 100%. This is because the job has lower priority, but is not suspended. If you use qstat, you'll see that the job's state is "U", because PBS has marked the job as "suspended". Local activity on the machine will have higher priority.

3.7 Restricting User Access to Execution Hosts

PBS provides a facility to prevent users from using machines controlled by PBS except by submitting jobs. You can turn this feature on using the $restrict_user MOM directive. This uses the $restrict_user_exceptions and $restrict_user_maxsysid directives. This can be set up vnode by vnode so that a user requesting exclusive access to a set of vnodes will be guaranteed that no other user will be able to use the nodes assigned to his job, or a user requesting non-exclusive access to a set of nodes will be guaranteed that no access will be allowed to the nodes except through PBS. Also, a privileged user can be allowed access to the complex such that they can login to a vnode without having a job active, or an abusive user can be denied access to the complex nodes. The administrator can find out when users try to circumvent a policy of using PBS to access nodes. In addition, you can ensure that application timings will be reproducible on a complex controlled by PBS. The log level for messages concerning restricting users is PBSE_SYSTEM (0002).

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 logging level 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 user names.

Examples:

Turn access restriction on for a given node:
$restrict_user on

Limit the users affected to those with a user ID greater than 500:
$restrict_user_maxsysid 500

Exempt specific users from the restriction:
$restrict_user_exceptions userA, userB, userC

3.8 Resource 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), and by periodically summing the usage recorded in the /proc entries. Note: enforcement is (1) site optional (one must add "$enforce mem" to the MOM's config file), and (2) only happens if the job requests a limit (via "mem=..." in the qsub parameters).

Job resource limits can be enforced for single-vnode jobs, or for multi-vnode jobs 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 Enforcement on UNIX” on page 138. The ncpus limit can be adjusted in several ways; for a discussion see “Job NCPUS Limit Enforcement” on page 140.

Table 5: Resource Limit Enforcement

Limit	What determines when limit is enforced	Scope of limit	Enforcement method
file size	automatically	per-process	setrlimit()
pvmem	automatically	per-process	setrlimit()
pmem	automatically	per-process	setrlimit()
pcput	automatically	per-process	setrlimit()
cput	automatically	job-wide	MOM poll
walltime	automatically	job-wide	MOM poll
mem	if `$enforce mem` in MOM’s config	job-wide	MOM poll

Table 5: Resource Limit Enforcement

Limit	What determines when limit is enforced	Scope of limit	Enforcement method
ncpus	if `$enforce cpuaverage`, `$enforce cpuburst`, or both, in MOM's config. See "Job NCPUS Limit Enforcement" on page 140.	job-wide	MOM poll

3.8.1 Job Memory Limit Enforcement on UNIX

Enforcement of `mem` resource usage is available on all UNIX platforms, but not Windows.
To enforce `mem` resource usage, put `$enforce mem` into MOM's config file. Enforcement is off by default.

The `mem` resource can be enforced at both the job level and the vnode level. The job level will be the smaller of a job-wide resource request and the sum of that for all chunks. The vnode level is the sum for all chunks on that node.

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 "Configuring MOM's Polling Cycle" on page 122.

Per-process limits are enforced by the operating system kernel. PBS calls the kernel call setrlimit() to set the limit for the top process (the shell) and any process started by the shell inherits those limits.

If a user submits a job with a job limit, but not per-process limits (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 qsub -lcput=10:00

a) There are two CPU-intensive processes which use 5:01 each.
The job will be killed by PBS for exceeding the cput limit.
5:01 + 5:01 is greater than 10:00.

b) There is one CPU-intensive process which uses 10:01.
It is very likely that the kernel will detect it first.

c) 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 `pmem` limit or without `pmem` and with a `mem` limit, PBS uses the `setrlimit`(2) call to set the limit. For most operating systems, `setrlimit`() is called with `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.

The following table shows which OS resource limits can be used by each operating system.

Table 6: RLIMIT Usage in PBS Professional

OS	file	mem/pmem	vmem/ pvmem	cput/pcput
AIX	RLIMIT _FSIZE	RLIMIT_RSS	RLIMIT_ DATA RLIMIT_ STACK	RLIMIT_CPU
HP-UX	RLIMIT _FSIZE	RLIMIT_RSS	RLIMIT_ AS	RLIMIT_CPU
Linux	RLIMIT _FSIZE	RLIMIT_RSS	RLIMIT_ AS	RLIMIT_CPU
SunOS	RLIMIT _FSIZE	RLIMIT_DATA RLIMIT_STACK	RLIMIT_ VMEM	RLIMIT_CPU

Table 6: RLIMIT Usage in PBS Professional

OS	file	mem/pmem	vmem/ pvmem	cput/pcput
Super-UX	RLIMIT_FSIZE	RLIMIT_UMEM RLIMIT_DATA RLIMIT_STACK	ignored	RLIMIT_CPU

For mem/pmem, the limit is set to the smaller of the two. For vmem/pvmem, the limit is set to the smaller of the two. Note that RLIMIT_RSS, RLIMIT_UMEM, and RLIMIT_VMEM are not standardized (i.e. do not appear in the The Open Group Base Specifications Issue 6).

3.8.1.1 Sun Solaris-specific Memory Enforcement

Solaris does not support `RLIMIT_RSS`, but instead has `RLIMIT_DATA` and `RLIMIT_STACK`, which are hard limits. On Solaris or another Open Group standards-compliant OS, a `malloc`() call that exceeds the limit will return `NULL`. This behavior is different from other operating systems and may result in the program (such as a user's application) receiving a `SIGSEGV` signal.

3.8.1.2 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 mem_exclusive would prevent two CPU sets from sharing the memory). Essentially, PBS enforces the equivalent of mem_exclusive by itself.

3.8.2 Job NCPUS Limit Enforcement

Enforcement of the `ncpus` limit (number of CPUs used) is available on all platforms. The **`ncpus`** limit can be enforced using average CPU usage, burst CPU usage, or both. By default, enforcement of the ncpus limit is off. See "$enforce <limit>" on page 114.

3.8.2.1 Average CPU Usage Enforcement

To enforce average CPU usage, put "`$enforce cpuaverage`" in MOM's 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 7: Variables Used in Average CPU Usage

Variable	Type	Description	Default
cpuaverage	Boolean	If present (=true), MOM enforces ncpus when the average CPU usage over the job's lifetime usage is greater than the specified limit.	false
average_trialperiod	integer	Modifies cpuaverage. Minimum job walltime before enforcement begins. Seconds.	120
average_percent_over	integer	Modifies cpuaverage. Percentage by which the job may exceed ncpus limit.	50
average_cpufactor	float	Modifies cpuaverage. ncpus limit is multiplied by this factor to produce actual limit.	1.025

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:

(cput / walltime) > (ncpus * average_cpufactor + average_percent_over / 100)

3.8.2.2 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 8: Variables Used in CPU Burst

Variable	Type	Description	Default
cpuburst	Boolean	If present (=true), MOM enforces ncpus when CPU burst usage exceeds specified limit.	false
delta_percent_over	integer	Modifies cpuburst. Percentage over limit to be allowed.	50
delta_cpufactor	float	Modifies cpuburst. ncpus limit is multiplied by this factor to produce actual limit.	1.5
delta_weightup	float	Modifies cpuburst. Weighting factor for smoothing burst usage when average is increasing.	0.4
delta_weightdown	float	Modifies cpuburst. Weighting factor for smoothing burst usage when average is decreasing.	0.1

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.

```
new_percent = change_in_cpu_time*100 / change_in_walltime
weight = delta_weight[up|down] * walltime/max_poll_period
new_cpupercent = (new_percent * weight) + (old_cpupercent *
    (1-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 > ((ncpus * 100 * delta_cpufactor) + delta_percent_over)

The following entries in MOM's config file turns 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
```

Cpuburst and cpuaverage information show up in MOM's log file, whether or not they has been configured in mom_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".

3.9 Configuring MOM for Machines with cpusets

There is an enhanced PBS MOM called pbs_mom.cpuset which is designed to manage a machine with cpusets. Using cpusets on the Altix requires the SGI ProPack library. See SGI's documentation for more information. The standard PBS MOM can also manage a machine with cpusets, but PBS and the jobs it manages will not create or otherwise make use of them.

3.9.1 Vnodes and cpusets

A cpuset is a list of CPUs and memory nodes managed by the OS. Processes executing within a cpuset are typically confined to use only the resources defined by the set. An Altix using `pbs_mom.cpuset` will present multiple vnodes to its server; these in turn are visible when using commands such as `pbsnodes`. Each of these vnodes is being managed by the one instance of `pbs_mom.cpuset`.

3.9.2 Rules for Creating cpusets

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.

3.9.3 Configuration Files for Multi-vnoded Machines

PBS uses three kinds of configuration files: the default configuration file described in "Syntax and Contents of Default Configuration File" on page 111, PBS reserved configuration files, which are created by PBS, and site-defined configuration files, described in "Syntax of Version 2 PBS

Reserved Configuration Files” on page 145.

The default configuration file lists MOM resources and initialization values. To change this file, you edit it directly.

Site-defined configuration files are used to make site-specific changes in vnode configuration. Instead of editing these directly, you create a local file and give it as an argument to the `pbs_mom -s insert` option, and PBS creates a new configuration file for you. See “Creation of Site-defined MOM Configuration Files” on page 109. Their syntax is called “version 2” in order to differentiate it from the syntax of the default configuration files. You can also remove a site-defined configuration file using the `pbs_mom -s remove` option.

PBS reserved files contain vnode configuration information. These are created by PBS. Any attempt to operate on them will result in an error.

You can list and view the PBS reserved configuration files and the site-defined configuration files using the `pbs_mom -s list` and `pbs_mom -s show` options.

Do not mix the configuration files or the syntax. Each type must use its own syntax, and contain its own type of information.

3.9.3.1 Creation of PBS Reserved Configuration Files

Any PBS reserved MOM configuration files are only created when PBS is started via the pbs start/stop script, not when the MOM is started with the pbs_mom command. Therefore, 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 “<path-to-script>/pbs start”, in order to re-create the PBS reserved MOM configuration files. The MOM daemon will normally be started by the PBS start/stop script.

3.9.3.2 Syntax of Version 2 PBS Reserved Configuration Files

These configuration files contain the configuration information for vnodes, including the resources available on those vnodes. They do not contain initialization values for MOM. The resources described in these configuration files can be set via `qmgr` and can be viewed using `pbsnodes -av`.

PBS reserved configuration files and site-defined configuration files use this syntax. Do not use this syntax for the default configuration file, and do not use the default configuration file's syntax to describe vnode information. For information about vnodes, see section 2.8 “Vnodes: Virtual Nodes” on page 48.

Any configuration file containing vnode-specific assignments must begin with this line:

$configversion 2

The format a file containing vnode information is:

<ID> : <ATTRNAME> = <ATTRVAL>

where

<ID>	sequence of characters not including a colon (":")
<ATTRNAME>	sequence of characters beginning with alphabetics or numerics, which can contain underscore ("_") and dash ("-")
<ATTRVAL>	sequence of characters not including an equal sign ("=")

The colon and equal sign may be surrounded by white space.

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. Vnode attributes are listed in section 2.9 “Vnode Configuration Attributes” on page 53.

3.10 Configuring MOM on an Altix

The configuration information for the Altix in this book is in three sections. The information common to all MOMs applies to the Altix; see section 3.2 “MOM Configuration Files” on page 109. For information on ProPack 4 and 5, see section 3.10 “Configuring MOM on an Altix” on page 147. For information on the Altix ICE/XE, see section 3.11 “Configuring MOM on SGI ICE with ProPack 5” on page 154.

To verify which CPUs are included in a cpuset created by PBS, on ProPack 4/5, use:

```
cpuset -d <set name> | egrep cpus
```

This will work either from within a job or not.

The alt_id returned by MOM has the form `cpuset=<name>`. <name> is the name of the cpuset, which is the $PBS_JOBID.

A cpusetted machine can have a "boot cpuset" defined by the administrator. 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 will contain CPU 0. By default, the PBS MOM will not use the boot 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.

The MOM excludes from its use all CPUs in sets not belonging to PBS. The way to reserve some for other uses is to create a boot CPU set.

In order to use `pbs_mom.cpuset` on an Altix, you will 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. Each Altix may have a different topology, depending on how it is wired. The PBS startup script creates the vnode definitions file for ProPack 4 and greater if it detects that pbs_mom.cpuset has been copied to pbs_mom.

The cpuset hierarchy has changed for version 8.0 and later. There are no directories under /PBSPro for shared or suspended cpusets.

On a suspend request, the cpuset MOM will move the processes to the global cpuset, then restore them later upon restart.

When PBS Professional creates job cpusets, it does not set the CPU or memory exclusive flags. PBS manages the exclusivity on these cpusets.

3.10.1 Configuring MOM for an Altix Running ProPack 4/5

On an Altix running ProPack 4/5, the vnode definitions file is generated automatically by PBS. The MOM includes routers automatically when she generates the file. There is a script which can be modified to produce different vnode definitions. The script is $PBS_EXEC/lib/init.d/sgigenvnodelist.awk. This script is designed to be modified by the PBS administrator. It is an alternative to using pbs_mom -s to insert changed vnode definitions.

3.10.2 Altix-Specific Configuration Parameters in Default MOM Configuration File

3.10.2.1 Static Resources for Altix Running ProPack 4 or 5

cpuset_create_flags <flags>

CPUSET_CPU_EXCLUSIVE | 0
Default: CPUSET_CPU_EXCLUSIVE

cpuset_destroy_delay <delay>

MOM will wait `delay` seconds before destroying a cpuset of a just-completed job. This allows processes time to finish. Default: 0. Integer. For example,

```
cpuset_destroy_delay 10
```

3.10.2.2 Initialization Values for Altix Running ProPack 4 or 5

pbs_accounting_workload_mgmt <value>

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.

3.10.2.3 Switching From Standard MOM to Cpusetted MOM on Altix

If you switch from the standard MOM to the cpusetted MOM, you'll need to 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. You will also need to delete the non-cpuset MOM's vnode and create a new one. Do not set mem, vmem, ncpus or sharing on the new vnode. Here are the steps:

1 Using qmgr, delete the vnode run by the MOM to be switched:

```
qmgr: delete node foo
```

2 Stop PBS:

```
/etc/init.d/pbs stop
```

3 Change directory to PBS_EXEC

4 Copy cpusetted MOM to MOM:

```
cp pbs_mom.cpuset pbs_mom
```

5 Start PBS:

```
/etc/init.c/pbs start
```

6 Using qmgr, create natural vnode :

```
qmgr: create node foo
```

3.10.2.4 Switching From Cpusetted 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.

Remove your own vnode definitions files. List them:

```
pbs_mom -s list
```

Remove each file you added:

```
pbs_mom -s remove <scriptname>
```

Add new configuration files with any information you need:

```
pbs_mom -s insert <new scriptname>
```

Then stop and start the mom to get the changes to take effect.

3.10.3 MOM Configuration Options on the SGI ICE with ProPack 5

3.10.3.1 Static Resources for ICE Running ProPack 5

cpuset_create_flags <flags>
CPUSET_CPU_EXCLUSIVE | 0
Default: 0

3.10.4 Configuring MOM for Comprehensive System Accounting

3.10.4.1 Requirements for CSA

Using CSA requires the version of pbs_mom.cpuset that is built with CSA enabled. CSA can be used on SGI Altix machines running SGI's ProPack 4 or greater, and having library (not system) call interfaces to the kernel's job container and CSA facilities. Both the Linux job container facility and CSA support must either be built into the kernel or available as loadable modules.

For information on getting Linux job container software configured and functioning, go to http://www.ciemat.es/informatica/gsc/perfdoc/007-4413-003/sgi_html/index.html and see "Linux Resource Administration Guide", subsection "Linux Kernel Jobs".

See the Release Notes for information on which versions of ProPack provide support for CSA with PBS.

If CSA is enabled, the PBS user 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 workload management is enabled, the kernel will write 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.

There are two pbs_mom daemons for the Altix, one for cpusets and the standard daemon.
The downloadable CSA-enabled PBS binaries for the Altix are built so that job container and CSA facilities are available in the kernel, so that both CSA user job accounting and CSA workload management accounting are available in both of the pbs_mom daemons.

In order for CSA user job accounting and workload management accounting requests to be acted on by the kernel, the administrator needs 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
```

To set CSA_START to “on”, use the command:

```
csaswitch -c on -n csa
```

To set WKMG_START to “on”, use:

```
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 manpage for csaswitch and how it is used in the /etc/init.d/csa startup script.

3.10.4.2 Configuration for CSA

If MOM is configured for CSA support, MOM can issue CSA workload management record requests to the kernel. To configure MOM for CSA support, modify $PBS_HOME/mom_priv/config, by adding a line for the parameter `pbs_accounting_workload_mgmt`. Set this parameter to “on”/”true”/”1” to enable CSA support, and “off”/”false”/”0” to disable it. If the parameter is absent, CSA support is enabled by default.

After modifying the MOM config file, either restart pbs_mom or send it SIGHUP.

For information on SGI Job Containers, see “SGI Job Container / Limits Support” on page 329.

3.10.5 Troubleshooting ProPack 4/5 cpusets

The ProPack4/5 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, one may see

> "/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 will be necessary to fix this problem. However, it is possible that if pbs_mom is started with the -p flag, one 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 `cpuset(1)` utility to test it:

```
# cpuset -s /PBSPro -r | while read set
    do
      cpuset -d $set > /dev/null
    done
```

If cpuset detects no problems, no output is expected. If a problem is seen, expect output of the form

> cpuset </badset> query failed
> /badset: Unknown error

In this case, try to remove the offending cpuset by hand, using the cpuset(1) utility,

```
# cpuset -x badset
cpuset <badset> removed.
```

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,

```
# cpuset -x nonempty
cpuset <nonempty> remove failed
/nonempty: Device or resource busy
```

first remove any CPU sets it contains:

```
# 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.
```

Note that output is previous output, reversed.

If the set has processes that are still attached,

```
# cpuset -x busy
cpuset <busy> remove failed
/busy: Device or resource busy
```

one can choose to either kill off the processes,

```
# 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, one may also try to remove it with rmdir(1), provided one takes care to prepend the cpuset file system mount point first. For example,

```
# mount | egrep cpuset
cpuset on /dev/cpuset type cpuset (rw)
# find /dev/cpuset/nonempty -type d -print |
     tac | while read set
   do
      rmdir $set
   done
```

3.11 Configuring MOM on SGI ICE with ProPack 5

You can choose whether or not to use cpusets on the SGI ICE. To use cpusets, install pbs_mom.cpuset; to use the SGI ICE without cpusets, install pbs_mom.standard. To install pbs_mom.cpuset, see section 4.8.1 "Installing MOM with SGI cpuset Support" on page 51 in the PBS Professional Installation & Upgrade Guide. If you do not take these steps, pbs_mom.standard is installed.

If you are running the cpuset MOM, the `init.d/pbs` script will configure one vnode per MOM. This enables cpusets and sets sharing to default_shared.

To provide the maximum number of available CPUs on a small node, make sure that the file `/etc/sgi-compute-node-release` is present. This way, on installation the pbs_habitat script will add a "`cpuset_create_flags 0`" to Mom's config file.

In order to exclude CPU 0, change the MOM configuration file line to

```
cpuset_create_flags CPUSET_CPU_EXCLUSIVE
```

This flag controls only whether CPU 0 is included in the PBS cpuset.

There is only one logical memory pool available per node on the SGI ICE. If, at startup, MOM finds:

- any CPU in an existing, non-root, non-PBS cpuset
- CPU 0 has been excluded as above

MOM will

- Exclude that CPU from the top set `/dev/cpuset/PBSPro`
- Create the top set with `mem_exclusive` set to false

Otherwise, the top set is created using all CPUs and with `mem_exclusive` set to true.

3.12 MOM Globus Configuration

For the optional Globus MOM, the same configuration mechanism applies as with the regular MOM except only three initiation value parameters are applicable: `$clienthost`, `$restricted`, `$logevent`. For details, see the description of these configuration parameters earlier in this chapter. Examples of different MOM configurations are included in Chapter 12 "Example Configurations" on page 399.

Chapter 4

Configuring the Scheduler

The Scheduler implements the local site policy determining which jobs are run, and on what resources. This chapter discusses the default configuration created in the installation process, and describes the full list of tunable parameters available.

4.1 New Scheduler Features

4.1.1 Extension to Tunable Formula

The tunable formula has been extended to include division, parentheses, exponentiation an d unary plus and minus. See section 4.7.2 “Tunable Formula for Computing Job Priorities” on page 194.

4.1.2 Eligible Wait Time for Jobs

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 tunable formula. Jobs have two new attributes, eligible_time and accrue_type, which indicates what kind of wait time the job is accruing. See section 4.7.3 "Eligible Wait Time for Jobs" on page 200.

4.1.3 Standing Reservation of Resources

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 "Advance and Standing Reservations" on page 204.

4.2 Scheduling Policy

The scheduler runs just one scheduling policy, which you can define. You can define placement sets and user and group resource and job limits, etc. However, you cannot have two different scheduling policies on two different queues or partitions. Whatever is set in the scheduler's configuration file applies to all queues or partitions.

4.2.1 Default Scheduler Configuration

The scheduler provides a wide range of scheduling policies. It provides the ability to sort the jobs in several different ways, in addition to FIFO order, such as on user and group priority, fairshare, and preemption. As distributed, it is configured with the following options (which are described in detail below).

1. Specific system resources are checked to make sure they are available: `mem` (memory requested), `ncpus` (number of CPUs requested), `arch` (architecture requested), **`host`**, and **`vnode`**.

2. Queues are sorted into descending order using the queue `priority` attribute to determine the order in which jobs are to be considered. 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, queues are ordered randomly.

3. Jobs within queues of priority `preempt_queue_prio` (default 150) or higher will preempt jobs in lower priority queues.

4. The jobs within each queue are sorted into ascending order of requested CPU time (`cput`). The shortest job is placed first.

5. Jobs that have waited to run for the amount of time specified in `max_starve` are *starving*. `max_starve` defaults to 24 hours. Starving jobs are given higher priority.

6. Any queue whose name starts with "`ded`" is treated as a dedicated time queue (see discussion below). A sample dedicated time file (*PBS_HOME*/`sched_priv/dedicated_time`) is included in the installation.

7. Primetime is set to 6:00 AM - 5:30 PM. Any holiday is considered non-prime. 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.

8. In addition, the Scheduler utilizes the following parameters and resources in making scheduling decisions:

Table 1:

Object	Attribute/Resource	Comparison
server, queue & vnode	resources_available	>= resources requested by job
server, queue & vnode	max_running	>= number of jobs running
server, queue & vnode	max_user_run	>= number of jobs running for a user
server, queue & vnode	max_group_run	>= number of jobs running for a group
server & queue	max_group_res	>= usage of specified resource by group
server & queue	max_user_res	>= usage of specified resource by user
server & queue	max_user_res_soft	>= usage of specified resource by user (see section 2.5 “Hard and Soft Limits” on page 16) Not enabled by default.
server & queue	max_user_run_soft	>= maximum running jobs for a user (see section 2.5 “Hard and Soft Limits” on page 16) Not enabled by default.
server & queue	max_group_res_soft	>= usage of specified resource by group (see section 2.5 “Hard and Soft Limits” on page 16) Not enabled by default.

Table 1:

Object	Attribute/Resource	Comparison
server & queue	max_group_run_soft	>= maximum running jobs for a group (see section 2.5 “Hard and Soft Limits” on page 16) Not enabled by default.
queue	started	= true
queue	queue_type	= execution
job	job_state	= queued / suspended
node	loadave	Boolean in sched_config. Used with max_load and ideal_load. When the loadave is above max_load, that node is marked “busy”. The scheduler won’t place jobs on a node marked “busy”. When the loadave drops below ideal_load, the “busy” mark is removed. Consult your OS documentation to determine values that make sense. Default: not enabled.
node	arch	= type requested by job
node	host	= name requested by job

4.2.2 Jobs that Can Never Run

A job that can never run 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 error message “resource request is impossible to solve: job will never run” is printed in the scheduler’s log file. The scheduler then examines the next job in line to be the most deserving job.

The scheduler only determines if a job will never run if backfilling is used. If backfilling is turned off, then the scheduler won't determine if a job will ever run or not. It just decides it can't run now.

4.3 Scheduler Configuration Parameters

To tune the behavior of the scheduler, change directory to *PBS_HOME/* sched_priv and edit the scheduling policy configuration file sched_config. This file controls the scheduling policy (the order in which jobs run). The format of the sched_config file is:

    name: value [prime | non_prime | all | none]

name cannot contain any whitespace, but *value* may if the string is double-quoted. *value* can be: true | false | number | string. Any line starting with a "#" is a comment, and is ignored. The third field allows you to specify that the setting is to apply during primetime, non-primetime, or all the time. A blank third field is equivalent to "all" which is both prime- and non-primetime. Note that the *value* and *all* are case-sensitive, but common cases are accepted, e.g. "TRUE", "True", and "true".

Important: Note that some Scheduler parameters have been deprecated, either due to new features replacing the old functionality, or due to automatic detection and configuration. Such deprecated parameters are no longer supported, and should ***not*** be used as they may cause conflicts with other parameters.

The available scheduling options, and the default values, are as follows.

backfill
: Boolean. If this is set to "True", the scheduler will attempt to schedule smaller jobs around starving jobs and 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 starving jobs will be considered first in the set to fit around the most starving job. For starving jobs, it only has an effect if the parameter "help_starving_jobs" is true. If backfill is "False", the scheduler will idle the system to run starving jobs. Can be used with strict_ordering.
Default: true all

backfill_prime
: boolean: Directs the Scheduler not to run jobs which will overlap the boundary between primetime and non-primetime. This assures that jobs restricted to running in either primetime or non-primetime can start as soon as the time boundary happens. See also prime_spill, **prime_exempt_anytime_queues**.
Default: false all

by_queue
: boolean. If set to true, jobs are run first from the first queue until that queue is empty, then the next queue, and so on. If **sort_queues** is set to true, queues are ordered highest-priority first. If **by_queue** is set to false, all jobs are treated as if they are in one large queue. The **by_queue** attribute is overridden by the **round_robin** attribute when **round_robin** is set to true. See section 4.9 “How Queues are Ordered” on page 208.

Default: true all

cpus_per_ssinode
: Deprecated. Such configuration now occurs automatically.

dedicated_prefix
: string: Queue names with this prefix will be treated as dedicated queues, meaning jobs in that queue will only be considered for execution if the system is in dedicated time as specified in the configuration file *PBS_HOME*/sched_priv/dedicated_time. See also section 4.10 “Defining Dedicated Time” on page 208.
Default: ded

fair_share
: boolean: This will enable the fairshare algorithm. It will also turn on usage collecting and jobs will be selected based on a function of their recent usage

and priority (shares). See also section 4.16 “Using Fairshare” on page 217.
Default: `false all`

`fairshare_entity`
string: Specifies the “entity” for which fairshare usage data will be collected. Can be “euser”, “egroup”, “Account_Name”, “queue”, or “egroup:euser”.
Default: `euser`

`fairshare_enforce_no_shares`
boolean: If this option is enabled, jobs whose entity has zero shares will never run. Requires `fair_share` to be enabled.
Default: `false`

`fairshare_usage_res`
string: Specifies the resource to collect and use in fairshare calculations and can be any valid PBS resource, including user-defined resources. See also section 4.16.5 “Tracking Resource Usage” on page 222. A special case resource is the exact string “ncpus*walltime”. The number of CPUs used is multiplied by the walltime in seconds used by the job to determine the usage.
Default: “`cput`”.

`half_life`
time: The half life for fairshare usage; after the amount of time specified, the fairshare usage will be halved. Requires that `fair_share` be enabled. See also section 4.16 “Using Fairshare” on page 217.
Default: `24:00:00`

`help_starving_jobs`
boolean: Setting this option will enable starving jobs 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.
Default: `true all`

`job_sort_key` string: Selects how the jobs should be sorted. `job_sort_key` can be used to sort by either resources or by special case sorting routines. Multiple `job_sort_key` entries can be used, in which case the first entry will be the primary sort key, the second will be used to sort equivalent items from the first sort, etc. The `HIGH` option implies descending sorting, `LOW` implies ascending. See example for details.

This attribute is overridden by the **`job_sort_formula`** attribute. If both are set, **`job_sort_key`** is ignored and an error message is printed.

Syntax: `job_sort_key: "PBS_resource HIGH|LOW"`
Default: "`cput low`"

There are three special case sorting routines, that can be used instead of a specific PBS resource:

Table 2:

Special Sort	Description
fair_share_perc HIGH	Sort based on the values in the resource group file. This should only be used if strict priority sorting is needed. **Do not enable `fair_share_perc` sorting if using the `fair_share` scheduling option.** (This option was previously named "`fair_share`" in the deprecated `sort_by` parameter). See also section 4.17 "Enabling Strict Priority" on page 226
job_priority HIGH|LOW	Sort jobs by the job `priority` attribute regardless of job owner. (The `priority` attribute can be set during job submission via the "`-p`" option to the `qsub` command, as discussed in the **PBS Professional User's Guide**.)
preempt_priority HIGH	Sort jobs by preemption priority. Recommended that this be used when soft user limits are used. Also recommended that this be the primary sort key.
sort_priority HIGH|LOW	**Deprecated**. See **`job_priority`**, above.

The following example illustrates using resources as a sorting parameter. Note that for each, you need to specify HIGH (descending) or LOW (ascending). Also, note that *resources* must be a quoted string.

```
job_sort_key: "ncpus HIGH" all
job_sort_key: "mem LOW" prime
```

`key` Deprecated. Use `job_sort_key`.

`load_balancing`

boolean: If set, the Scheduler will balance the computational load of single-host jobs across a complex. The load balancing takes into consideration the load on each host as well as all resources specified in the "resource" list. See `smp_cluster_dist`, and section 4.13 "Enabling Load Balancing" on page 213. Load balancing can result in overloaded CPUs.
Default: `false all`

`load_balancing_rr`

Deprecated. To duplicate this setting, enable `load_balancing` and set `smp_cluster_dist` to `round_robin`. See also section 4.13 "Enabling Load Balancing" on page 213.

`log_filter`

integer: Defines which event types to keep out of the scheduler's logfile. The value should be set to the bitwise OR of the event classes which should be filtered. (A value of 0 specifies maximum logging.) See also section 6.17 "Use and Maintenance of Logfiles" on page 353.
Default: `1280` (DEBUG2 & DEBUG3)

`max_starve`

time: The amount of time before a job is considered starving. This variable is used only if `help_starving_jobs` is set.
Format: HH:MM:SS
Default: `24:00:00`

`mem_per_ssinode`

Deprecated. Such configuration now occurs automatically.

`mom_resources`

string: This option is used to query the MOMs to set the value of `resources_available.RES` where RES is a site-defined resource. Each MOM is queried with the resource name and the return value is used to replace `resources_available.RES` on that vnode. On a multi-vnoded machine with a natural vnode, all vnodes will share anything set in `mom_resources`.

`node_sort_key`

string: Defines sorting on resource values on vnodes. Resource must be numerical, for example, `long` or `float`.
Syntax:
node_sort_key: “<resource>|job_priority \
HIGH|LOW”
node_sort_key: “<resource> HIGH|LOW \
total|assigned|unused”
“total”: Use the resources_available value.
“assigned”: Use the resources_assigned value.
“unused”: Use the value given by resources_available - resources_assigned.
See section 4.6.8.1 “Sorting Vnodes with node_sort_key” on page 182.
Note that up to 20 `node_sort_key` entries can be used, in which case the first entry will be the primary sort key, the second will be used to sort equivalent items from the first sort, etc.
Default:
node_sort_key: “job_priority HIGH”

`nonprimetime_prefix`

string: Queue names which start with this prefix will be treated as non-primetime queues. Jobs within these queues will only run during non-primetime. Primetime and non-primetime are defined in the `holidays` file. See also “Defining Primetime and Holidays” on page 209.
Default: `np_`

`peer_queue`

string: Defines the mapping of a remote queue to a local queue for Peer Scheduling. Maximum number is 50 peer queues per scheduler. For details, see section 4.18 “Enabling Peer Scheduling” on page 226.
Default: unset

`preemptive_sched`

string: Enable job preemption. See section 4.15 “Enabling Preemptive Scheduling” on page 214 for details.
Default: `true all`

`preempt_checkpoint`

Deprecated. Add “C” to `preempt_order` parameter.

`preempt_fairshare`

Deprecated. Add “fairshare” to `preempt_prio` parameter.

`preempt_order`

quoted list: Defines the order of preemption methods which the Scheduler will use 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 usage is an ordering (SCR) optionally followed by a percentage of time remaining and another ordering. Note, this has to be a quoted list(“”).
Default: `SCR`

preempt_order: “SR”
or
preempt_order: “SCR 80 SC 50 S”

The first example above specifies that PBS should first attempt to use suspension to preempt a job, and if that is unsuccessful, then requeue the job. The second example says 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_prio` quoted list: Specifies the ordering of priority of different preemption levels. Two or more job types may be combined at the *same* priority level with a "+" between them (no whitespace). Comma-separated preemption levels are evaluated left to right, with each having lower priority than the preemption level preceding it. The table below lists the six preemption levels. Note that any level not specified in the `preempt_prio` list will be ignored.
Default: `"express_queue, normal_jobs"`

Table 3:

express_queue	Jobs in the preemption (e.g. "express") queue(s) preempt other jobs (see also `preempt_queue_prio`).
starving_jobs	When a job becomes starving it can preempt other jobs.
fairshare	When the entity owning a job exceeds its fairshare limit.
queue_softlimits	Jobs which are over their queue soft limits
server_softlimits	Jobs which are over their server soft limits
normal_jobs	The preemption level into which a job falls if it does not fit into any other specified level.

For example, the first line below states that starving jobs have the highest priority, then normal jobs, and jobs whose entities are over their fairshare limit are third highest. The second example shows that starving jobs whose entities are also over their fairshare limit are lower priority than normal jobs.

```
preempt_prio: "starving_jobs,
normal_jobs, fairshare"
```
or
```
preempt_prio: "normal_jobs,
starving_jobs+fairshare"
```

preempt_queue_prio
: integer: Specifies the minimum queue priority required for a queue to be classified as an express queue.
Default: 150

preempt_requeue
: Deprecated. Add an “R” to preempt_order parameter.

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, job will be that with longest running time. See “Preemption Ordering by Start Time” on page 217.

preempt_starving
: Deprecated. Add “starving_jobs” to preempt_prio parameter.

preempt_suspend
: Deprecated. Add an “S” to preempt_order parameter.

primetime_prefix
: string: Queue names starting with this prefix are treated as primetime queues. Jobs will only run in these queues during primetime. Primetime and non-primetime are defined in the holidays file. See also “Defining Primetime and Holidays” on page 209.
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-prime-

time/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. See also `backfill_prime, prime_spill`.
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 is only meaningful if `backfill_prime` is true. Also note that this option can be separately specified for prime- and non-primetime. See also `backfill_prime, prime_exempt_anytime_queues`.
Units: time.
Default: `00:00:00`

For example, the first setting below means that non-primetime jobs can spill into primetime by 1 hour. However the second setting means that jobs in either prime/non-prime can spill into the other by 1 hour.

prime_spill: 1:00:00 prime
or
prime_spill: 1:00:00 all

`resources`
string: Specifies those resources which are to be enforced when scheduling jobs. Vnode-level boolean resources are automatically enforced and do not need to be listed here. Limits are set by setting `resources_available.resourceName` on the Server objects (vnodes, queues, and servers). The Scheduler will consider numeric (integer or float) items as consumable resources and ensure that no more are assigned than are available (e.g. `ncpus` or `mem`). Any string resources will be compared using string comparisons (e.g. `arch`).

Default: "`ncpus, mem, arch, host, vnode`" (number CPUs, memory, architecture). If host is not

added to the resources line, when the user submits a job requesting a specific vnode in the following syntax:
qsub -l select=host=vnodeName
the job will run on any host.

`resource_unset_infinite`
Comma-delimited list of host-level resources. Resources in this list will be treated as infinite if they are unset. Cannot be set differently for prime-time and non-primetime. Default: empty list.

Example: resource_unset_infinite: “vmem, foo_licenses”

`round_robin` boolean: If set to true, the scheduler will consider one job from the first queue, then one job from the second queue, and so on in a circular fashion. If **`sort_queues`** is set to true, the queues are ordered with the highest priority queue first. Each scheduling cycle starts with the same highest-priority queue, which will therefore get preferential treatment If **`round_robin`** is set to false, the scheduler will consider jobs according to the setting of the **`by_queue`** attribute.

When true, overrides the **`by_queue`** attribute.

Default: `false all`

`server_dyn_res`
string: Directs the Scheduler to replace the Server's `resources_available` values with new values returned by a site-specific external program. See section 5.5.1 “Dynamic Server-level Resources” on page 252 for details of usage.

`smp_cluster_dist`
string: Specifies how single-host jobs should be distributed to all hosts of the complex. Options are: `pack`, `round_robin`, and `lowest_load`.

`pack` means keep putting jobs onto one host until it is “full” and then move on to the next. `round_robin` is to put one job on each vnode in turn before cycling back to the first one. `lowest_load` means to put the job on the lowest-loaded host. See also section 4.12 “Configuring SMP Cluster Scheduling” on page 211, and section 4.13 “Enabling Load Balancing” on page 213. Default: `pack all`

`sort_by` **Deprecated**. Use `job_sort_key`.

`sort_queues` Boolean. When set to true, queues are sorted so that the highest priority queues are considered first. Queues are sorted by each queue’s priority attribute. The queues are sorted in a descending fashion, that is, a queue with priority 6 comes before a queue with priority 3. See section 4.9 “How Queues are Ordered” on page 208.

This is a prime option, which means it can be selectively applied to primetime or non-primetime.

Default: true ALL

`strict_fifo` **Deprecated**. Use strict_ordering.

`strict_ordering`
boolean: 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. The jobs are sorted at the server level, not the queue level. If a job due to run next cannot run, no job will run, unless backfilling is used. Jobs can be backfilled around the job that’s due to run next, if it is blocked. See section 4.19.1 “Enabling FIFO Scheduling with strict_ordering” on page 231. Default: false.
Example line in PBS_HOME/sched_priv/sched_config:

```
strict_ordering: true ALL
```

`sync_time` time: The amount of time between writing the fairshare usage data to disk. Requires `fair_share` to be enabled.
Default: `1:00:00`

`unknown_shares`
integer: The number of shares for the “unknown” group. These shares determine the portion of a resource to be allotted to that group via fairshare. Requires `fair_share` to be enabled. See section 4.16 “Using Fairshare” on page 217 for information on how to use fairshare.
The “unknown” group gets 0 shares unless set.

4.4 Scheduler Attributes

Scheduler attributes can be read only by the PBS Manager or Operator. All scheduler attributes are read-only.

`pbs_version` The version of PBS for this scheduler. Available only to Manager/Operator.

`sched_host` The hostname of the machine on which the scheduler runs. Available only to Manager/Operator.

4.5 How Jobs are Placed on Vnodes

Placement sets allow the administrator to group vnodes into useful sets, and have multi-vnode jobs run in one set. For example, it makes the most sense to run a job on vnodes that are all connected to the same high-speed switch. PBS places each job on one or more vnodes according to the job’s resource request, whether and how the vnodes have been grouped, and whether the vnodes can be shared. For more on sharing, see “sharing” on page 55.

Using placement sets, vnodes are partitioned according to the value of one or more resources. These resources are listed in the node_group_key

attribute. Grouping nodes is enabled by setting `node_group_enable` to True. If you use the server's node_group_key, the resulting groups apply to all of the jobs in the complex. If you use a queue's node_group_key, only jobs in that queue will have those groups applied to them.
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.

When the partitioning is done according to the values of more than one resource, that is, node_group_key lists more than one resource, the resulting groups are called placement sets. In placement sets, a vnode may belong to more than one set. For example, if a given vnode is on switch S1 but not switch S2 and router R1, it can belong to the set of vnodes that all share resources_available.switch=S1 and also to the set that all share resources_available.router=R1. It will not be in the set that all share resources_available.switch=S2. Each placement set is defined by the value of exactly one resource, not a combination of resources. A series of placement sets is created according to the values of a 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 these three values, then there will be three placement sets in the series. All of the placement sets defined by all of the resources in `node_group_key` are called a placement pool.

PBS will attempt to place each job in the smallest possible group or set that is appropriate for the job.

4.5.1 Placing Jobs in Reservations

When a reservation is created, it is created within a placement set. A job within a reservation runs within that placement set, but that is the only placement set considered for the job. Even if a reservation is so large that it spans placement sets, jobs in that reservation are not placed within those specific placement sets.

4.6 Placement Sets and Task Placement

Placement sets are the sets of vnodes within which pbs will try to place a job. PBS tries to determine which vnodes are connected (i.e. should be grouped together into one set), and the scheduler groups vnodes that share a placement value together in an effort to select which vnodes to assign to a job. The scheduler tries to put a job in the smallest appropriate placement set.

Placement sets are defined by string or multi-valued string resources chosen by the administrator. A placement set is the set of vnodes that share a value for a specific resource. A vnode can belong to more than one placement set defined by a multi-valued string resource. For example, if the resource is called “router”, and the vnode’s router resource is set to “router1, router2”, then the vnode will be in the placement set defined by router = router1 and the set defined by router = router2.

A placement pool is the collection of sets defined by one or more resources. So if we use only the resource called router, if the router resources on all the vnodes have some combination of router1 and router2, then there will be two placement sets in the router placement pool.

PBS may create default platform-dependent placement sets depending upon topology information. You can look for the resource names and values used for placement set names in the PBS-generated MOM configuration files. You can look for the names of the resources used to generate placement sets in the server’s pnames attribute.

4.6.1 Definitions

Task placement
: The process of choosing a set of vnodes to allocate to a job that will both satisfy the job's resource request (select and place specifications) and satisfy the configured Scheduling policy.

Placement Set
: A set of vnodes. Placement sets are used to improve task placement (optimizing to provide a “good fit”) by exposing information on system configuration and topology. Placement sets are defined using vnode-level resources of type multi-valued string. A single placement set is defined by one resource name and a single value; all vnodes in a placement set include an identical value for that specified resource. For example, assume vnodes have a resource named “switch”, which can have values

"A", "B", or "C": the set of vnodes which match "switch=B" is a placement set.

Placement Set Series A set of sets of vnodes. A placement set series is defined by one resource name and all its values. A placement set series is the set of placement sets where each set is defined by one value of the resource. If the resource takes on N values at the vnodes, then there are N sets in the series. For example, assume vnodes have a resource named "switch", which can have values "A", "B", or "C": there are three sets in the series. The first is defined by the value "A", where all the vnodes in that set have the value "A" for the resource "switch". The second set is defined by "B", and the third by "C".

Placement Pool A set of placement sets used for task placement. A placement pool is defined by one or more vnode-level resource names and the values of these resources on vnodes. In the example above, "switch" defines a placement pool of three placement sets. `node_group_key` defines a 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).

4.6.2 Configuring Placement Sets

Placement is turned on by setting:

```
qmgr> set server node_group_enable = True
qmgr> set server node_group_key = <resource
list>
```

For example, to create a placement pool for the resources vnodes, hosts, L2 and L3:

```
qmgr> set server node_group_key =
"vnode,host,L2,L3"
```

If there is a vnode level resource called "cbrick" set on the vnodes on the Altix, then the node_group_key should include cbrick too, i.e.,

```
qmgr> set server
node_group_key="vnode,host,cbrick,L2,L3"
```

4.6.3 Multihost Placement Sets

Placement pools and sets can span hosts. This applies to multi-vnode machines that have been partitioned into more than one system. To set up a multihost placement set, set a given resource on the vnodes for more than one host, then put that resource in the `node_group_key`. For example, create a string_array resource called “`span`” in the PBS_HOME/server_priv/resourcedef file:

span type=string_array

Add the resource “span” to node_group_key on the server or queue. Use qmgr to give it the same value on all the vnodes. You must write a script that sets the same value on each vnode that you want in your placement set.

4.6.4 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 a cpusetted Altix running ProPack 4 or 5, the placement information for cpusets is generated by PBS.

Node grouping allows vnodes to be in multiple placement sets. The string resource is a multi-valued string resource. Each value of the resource defines a different placement set. This creates a greater number of placement sets, and they may overlap (a vnode can be in more then one placement set). Not all placement sets have to contain the same number of vnodes.

4.6.5 Order of Precedence for Job Placement

Different placement pools can be defined complex-wide (server-level), and per-queue. A server-level placement pool is defined by setting the server's `node_group_key`. A queue-level placement pool is defined by setting the queue's `node_group_key`. Jobs can only define placement sets. 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 placement set. The scheduler uses the most specific placement pool for task placement for a job:

1 If there is a per-job placement set defined, it is used, otherwise,

2 If there is a per-queue placement pool defined for the queue the job is in, it is used, otherwise,

3 If there is a complex-wide placement pool defined, it is used, otherwise,

4 The placement pool consisting of one placement set of all vnodes is used.

This means that a job's place=group resource request overrides the sets defined by the queue's or server's node_group_key.

4.6.6 Defining Placement Sets

A placement pool is defined by one or more vnode-level resource names and the values of these resources on vnodes. This includes values that are unset or zero. For a single vnode-level resource RES which has `N` distinct values, `v1, ..., vN`, the placement set series defined by RES contains `N` sets of vnodes. Each set corresponds to one value of RES. For example, the placement set corresponding to RES and `v5` has the property that all vnodes in the set include `v5` in the value of RES. The placement pool defined by multiple resource names is simply the union of the placement pools defined by each individual resource name.

Server node_group_key attribute is an array of strings, e.g.,

```
Qmgr: set server
node_group_key="res1,res2, …, resN"
```

Queue-level node_group_key attribute (also an array of strings):

```
    Qmgr: set queue QNAME
node_group_key="res1, …resN"
```

The complex-wide placement pool is defined by all resource names listed in the server-level `node_group_key`. Similarly, per-queue placement pools are defined by the queue-level `node_group_key`. Either of these pools can be defined using multiple resource names. Per-job placement pools are defined by the single resource name given in the place directive (group=RES).

On a multi-vnoded system which is set up to do so, MOM sends the Server a list of resource names to be used by the Scheduler for placement set information.

4.6.7 Placement Sets Defined by Unset Resources

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.

4.6.8 Ordering and Choosing Placement Sets

The selected `node_group_key` defines the placement pool. The scheduler will order the placement sets in the placement pool.

The sets are sorted in this order:

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

The vnodes are sorted within a set in this order:

5 Vnodes sorted by node_sort_key if using node_sort_key (see "Sorting Vnodes with node_sort_key"below)

6 Order the vnodes are returned by pbs_statnode() if no node_sort_key. This is the default order the vnodes appear in the output of the command: “pbsnodes -a”.

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 dynamically fits. This ordering ensures the scheduler will use the smallest possible placement set in which the job will dynamically fit.

If a job cannot statically fit into any placement set in the placement pool, then the scheduler places the job in the placement set consisting of all vnodes. Note that if the user specifies **`-lplace=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.

4.6.8.1 Sorting Vnodes with `node_sort_key`

The vnodes within each placement set are sorted according to the `node_sort_key` option. The values sorted by `node_sort_key` must be numerical. The placement sets themselves are then ordered according to the criteria described in section 4.6.8 “Ordering and Choosing Placement Sets” on page 181. Up to 20 `node_sort_key` entries can be used, in which case the first entry will be the primary sort key, the second will be used to sort equivalent items from the first sort, etc.

Syntax:

```
node_sort_key: "<resource>|job_priority
HIGH|LOW"
node_sort_key: "<resource> HIGH|LOW
total|assigned|unused"
```

Specifying a <resource> such as mem or ncpus sorts vnodes by the resource specified.

total	Use the resources_available value.
assigned	Use the resources_assigned value.
unused	Use the value given by resources_available - resources_assigned.

If the third argument (total|assigned|unused) is not specified with a resource, “total” will be used. This provides backwards compatibility with previous releases.

Specifying `job_priority` sorts vnodes by their `priority` attribute, and cannot be used with a third argument (assigned|unused|total).

Default:

```
node_sort_key: “job_priority HIGH”
```

Examples
If we use

```
node_sort_key: “ncpus HIGH unused”
```

this will sort vnodes by the highest number of unused cpus.

If we use

```
node_sort_key: “mem HIGH assigned”
```

this will sort vnodes by the highest amount of memory assigned to vnodes.

The old “nodepack” behavior can be achieved by

```
node_sort_key: “ncpus low unused”
```

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. We use:

```
node_sort_key: “job_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.

4.6.8.2 Caveats

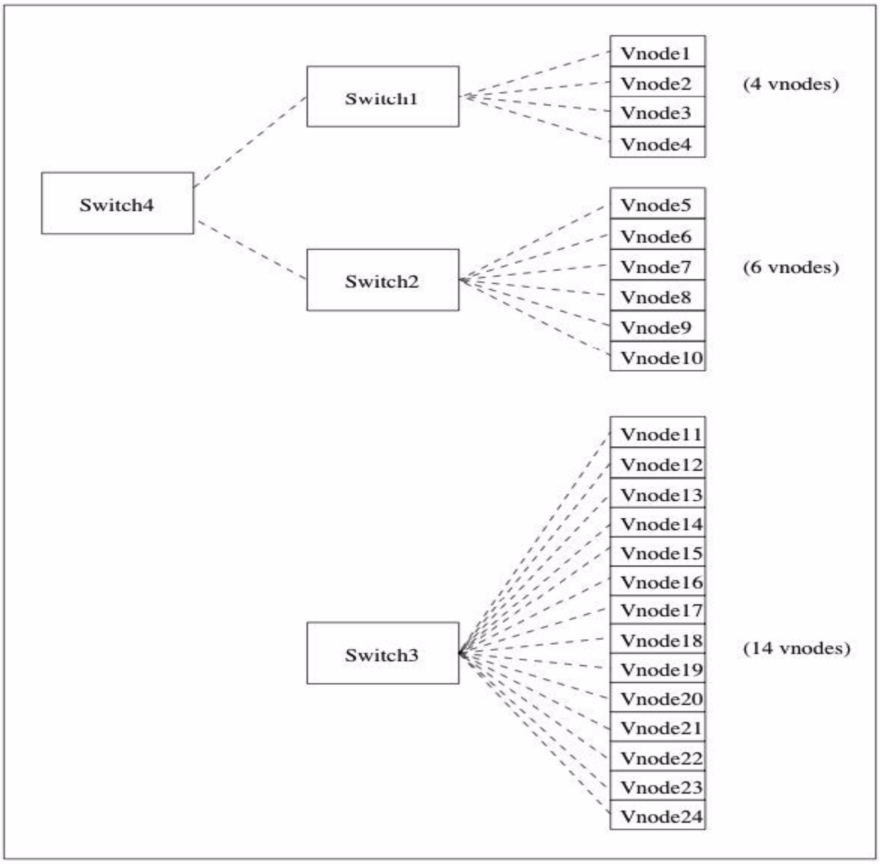
Sorting on a resource and using “unused” or “assigned” cannot be used with `load_balancing`. If both are used, load balancing will be disabled.

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”.

4.6.9 Placement Set Examples

4.6.9.1 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*:

```
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.

PBS will choose the smallest currently available set in which the job fits dynamically. If no set in which the job fits dynamically is available, it will wait any set to become available. If the job will not statically fit in any placement set, it will run in the placement set made up of all vnodes.

4.6.9.2 Examples of Configuring Placement Sets on an Altix

To define new placement sets on an Altix, you can either use the qmgr command or you can create a site-defined MOM configuration file. See section 3.2.1 “Creation of Site-defined MOM Configuration Files” on page 109 and the `-s script_options` option to `pbs_mom` in section 6.5.3.6 “Options to pbs_mom” on page 280.

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.

Step 1 Add the new resource to the server’s resourcedef file:

```
NewRes type=string
```

Step 2 Add "NewRes" to the server's node_group_key

```
qmgr> set server \
node_group_key="vnode,host,L2,L3,\
NewRes"
```

Step 3 Restart the server

Step 4 Add "NewRes" to the value of the `pnames` attribute for the natural vnode. Add a line like this to SetDefs:

```
altix3: resources_available.pnames
= \ L2,L3,NewRes
```

Step 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 = \
   <new set name>
```

All the vnodes in <new set name> should have lines of that form, with the same <new set name> value, in the new config file. That is, if vnodes A, B, and C comprise a placement set, add lines that specify the value of <new set name>. Here the value of <new set name> is “P”.

A: resources_available.NewRes = P
B: resources_available.NewRes = P
C: resources_available.NewRes = P

For each new placement set you define, use a different value for <new set name>.

Step 6 Add SetDefs and tell MOM to read it, to make a site-defined MOM configuration file NewConfig.

```
pbs_mom -s insert NewConfig SetDefs
pkill -HUP pbs_mom
```

You can define more than one placement set at a time. Next we will use NewRes2 and give it two values, so that we have two placement sets.

Step 1 Add the new resource to the server's resourcedef file:

```
NewRes2 type=string_array
```

Step 2 Add "NewRes2" to the server's node_group_key

```
qmgr> set server \
node_group_key="vnode,host,L2,L3, \
   NewRes2"
```

Step 3 Restart the server

Step 4 Add "NewRes2" to the value of the `pnames` attribute for the natural vnode. Add a line like this to SetDefs2:

```
altix3:  \
   resources_available.pnames = \
   L2,L3,NewRes2
```

Step 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 = \
   "<new set name1>, \
   <new set name2>"
```

Here, we'll put vnodes A, B and C into one placement set, and vnodes B, C and D into another.

```
A:  resources_available.NewRes2 = P
B:  resources_available.NewRes2 = \
    "P,Q"
C:  resources_available.NewRes2 = \
    "P,Q"
```

```
D:  resources_available.NewRes2 = Q
```

Step 6 Add SetDefs2 and tell MOM to read it, to make a site-defined MOM configuration file NewConfig.

```
pbs_mom -s insert NewConfig \
   SetDefs2
pkill -HUP pbs_mom
```

You can also use the qmgr command to set the values of the new resource on the vnodes.

```
Qmgr: set node B
resources_available.NewRes2="P,Q"
```

4.6.9.3 Example of 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**

Define the pool you want:

Qmgr: **s s node_group_key="cbrick, L2"**

If the vnodes look like this, from "pbsnodes -av ! egrep '(^[^]) | cbrick" or "pbsnodes -av ! egrep '(^[^]) | L2" :

```
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.
cbrick=cbrick1: {vnode1, vnode2}
cbrick=cbrick2: {vnode3, vnode4}
cbrick=cbrick3: {vnode5, vnode6}
cbrick=cbrick4: {vnode7, vnode8}
L2=A: {vnode1, vnode3, vnode5, vnode7}
L2=B: {vnode2, vnode4, vnode6, vnode8}

4.6.9.4 Colors Example

A placement pool is defined by two resources: `colorset1` and `colorset2`, by using "`node_group_key=colorset1,colorset2`". If a vnode has:
    resources_available.colorset1=blue, red
    resources_available.colorset2=green

The placement pool contains three placement sets. These are
    {resources_available.colorset1=blue}
    {resources_available.colorset1=red}
    {resources_available.colorset2=green}
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. `colorset=blue,red,green`.

Example: We have five vnodes `v1 - v5:`

v1 color=red host=mars
v2 color=red host=mars
v3 color=red host=venus
v4 color=blue host=mars
v5 color=blue host=mars

The placement pools are defined by
node_group_key=color

The resulting node groups would be: `{v1, v2, v3}, {v4, v5}`

4.6.9.5 Simple Node Grouping on Switch 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 node grouping so that jobs will be scheduled only onto the same switch.

First, create a new resource called “switch”. section 5.3 “Defining New Custom Resources” on page 241

Next, we need to enable node grouping 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}

4.6.10 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 vnodes with sharing=default_excl, jobs are assigned entire vnodes exclusively. For vnodes with sharing=default_shared, this causes a different allocation: unused memory on otherwise-allocated vnodes is allocated to the job. The exec_vnode attribute will show this allocation. Chunks are only placed on vnodes whose state is “free”.

On the Altix, the scheduler will share memory from a chunk even if all the cpus are used. It will first try to put a chunk entirely on one vnode. If it can, it'll run it there. If not, it'll break the chunk up across any vnode it can get resources from, even for small amounts of unused memory.

4.6.11 Reservations

The same rules about placement sets are used for reservations as are used for regular jobs.

4.6.12 Node Grouping

Node grouping is the same as one placement set series, where the placement sets are defined by one resource. This is also called complex-wide node grouping.

4.6.13 Non-backward-compatible Change in Node Grouping

Given the following example configuration:

```
        node1: switch=A
        node2: switch=A
        node3: switch=B
        node4: switch=B
        node5: 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 node grouping (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 “”.

4.7 Job Priorities in PBS Professional

There are various classes of default job priorities within PBS Professional, which can be enabled and combined based upon customer needs. The following table illustrates the inherent ranking of the defaults for these different classes of priorities. This is the ordering that the scheduler uses. A higher ranking class always takes precedence over lower ranking classes, but within a given class the jobs are ranked according to the attributes specific to that class. For example, since the Reservation class is the highest ranking class, jobs in that class will be run (if at all possible) before jobs from other classes. If a job qualifies for more than one category, it falls into the higher-ranked category. In the following table, higher-ranked classes are shown above lower-ranked.

Table 4: Classes of Job Priorities

Class	Description
Reservation	Jobs submitted to an advance or standing reservation, where resources are already reserved for the job.
Express	High-priority (“express” jobs). See discussion in section 4.15 “Enabling Preemptive Scheduling” on page 214.
Starving	Jobs that have waited longer than the starving job threshold. See the Scheduler configuration parameters `help_starving_jobs`, `max_starve`, and `backfill`. See the server’s `eligible_time_enable` attribute.
Suspended	Jobs that have been suspended by higher priority work.

Table 4: Classes of Job Priorities

Class	Description
round_robin or by_queue	Queue-based scheduling may affect order of jobs depending on whether these options are enabled.
job_sort_formula, fairshare, or job_sort_key	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`.

You can specify a formula for sorting jobs. This formula determines how jobs are sorted in the lowest ranked category in the table above. See section 4.7.2 “Tunable Formula for Computing Job Priorities” on page 194.

While the lowest category does sort jobs at the finest granularity, most of the work of sorting jobs is done in this category. The precedence of the categories cannot be changed.

4.7.1 Running Jobs in Submission Order

To run jobs in the order in which they were submitted, comment out the default `job_sort_key` in sched_priv/sched_config, and do not provide a job sorting formula in job_sort_formula. For example, to run jobs by queue priority, and then by submission order, with strict ordering and backfill, set the following:

```
by_queue: true
strict_odering: true
backfill: true
```

Give each queue a priority value.

4.7.2 Tunable Formula for Computing Job Priorities

You can choose to use a formula by which to sort jobs at the finest-granularity level. These levels are shown in the table “Classes of Job Priorities” on page 193. This formula will override both `job_sort_key` and fairshare for sorting at that level. You specify the formula in the server’s `job_sort_formula` attribute. If that attribute contains a formula, the scheduler will use it. If not, the scheduler computes job priorities according to fairshare, if fairshare is enabled. If neither is defined, the scheduler uses `job_sort_key`. When the scheduler sorts jobs according to the formula, it computes a priority for each job, where that priority is the value produced by the formula. Jobs with a higher value get higher priority.

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.

Once you set `job_sort_formula` via qmgr, it takes effect with the following scheduling cycle. The range for the formula is defined by the IEEE floating point standard for a double. 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. Variables are evaluated at the start of the scheduling cycle.

To set the `job_sort_formula` attribute, use the `qmgr` command.

```
Qmgr> s s job_sort_formula = "<formula>"
```

Only one formula is used to prioritize all jobs. If you change the formula at the server after some jobs are submitted but before others are submitted, all of the jobs will be ordered according to the same, later, formula, during the next scheduling cycle.
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.

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.

If an error is encountered while evaluating the formula, the formula evaluates to zero for that job, and the following message is logged at level DEBUG2: “1234.mars;Formula evaluation for job had an error. Zero value will be used”.

4.7.2.1 Formula Coefficients

The formula operates only on resources in the job’s Resource_List attribute. These are the job-level resources, and may have been explicitly requested, inherited, or summed from host-level resources. See section 2.11.2 “The Job’s Resource_List Attribute” on page 81. This means that all variables and coefficients in the formula must be resources that were either requested by the job or were inherited from the server or queue defaults. Formula coefficients are either custom numeric resources inherited by the job from the server or queue, or they are long integers or floats. Therefore you may need to create custom resources at the server or queue level to be used for formula coefficients.

Table 5: Terms in Tunable Formula

Terms		Allowable Value
Constants		NUM or NUM.NUM
Attribute values	queue_priority	Value of `priority` attribute for queue in which job resides
	job_priority	Value of the job’s `priority` attribute
	fair_share_perc	Percentage of fairshare tree for this job’s entity
	eligible_time	Amount of wait time job has accrued while waiting for resources
Resources		ncpus
		mem
		walltime
		cput
Custom numeric job-wide resources		Uses the amount requested, not the amount used. Must be of type long, float, or size. See section 5.1.1 “Custom Resource Formats” on page 238.

4.7.2.2 Modifying Coefficients For a Specific Job

Formula coefficients can be altered for each job by using the qalter command to change the value of that resource for that job. If a formula coefficient is a constant, it cannot be altered per-job.

4.7.2.3 Examples of Using the Tunable Formula

Examples of formulas:
Example 1: 10 * ncpus + 0.01*walltime + A*mem
Where "A" is a custom resource

Example 2: ncpus + 0.0001*mem

Example 3 : To change the formula on a job-by-job basis, alter the value of a resource in the job's Resource_List.RES. So if the formula is A *queue_priority + B*job_priority + C*ncpus + D*walltime, where A-D are custom numeric resources. 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: ncpus*mem

Example 5: Set via qmgr:

```
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 6:

```
Qmgr> s s job_sort_formula=ncpus
```

Example 7:

```
Qmgr> s s job_sort_formula='queue_priority +
ncpus'
```

Example 8:

```
Qmgr> s s job_sort_formula='5*job_priority + 
10*queue_priority'
```

4.7.2.4 Examples of Using Resource Permissions in Tunable Formula

For information on using resource permissions, see section 2.10.9 “Resource Flags for Resource Permissions” on page 71.

Example 1. You may want to create per-job coefficients in your tunable 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 *(Queue Priority) + B*(Job Class Priority) + C*(CPUs) + D*(Queue Wait Time)

Example 2. 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.

For example:

```
sched_priority = W_prio * wait_secs + P_prio 
* priority + ... +  special_priority
```

Here, special_priority is very large.

4.7.2.5 Units

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 from 1..N, size resources like mem are in kb, so 1gb is 1048576kb, and time 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:

Time resources:	seconds
Memory:	kb, so 1gb => 1048576kb
ncpus:	1..N

Example: 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: you are using gb of mem:

qmgr> **s s job_sort_formula=’1048576 * ncpus + 2 * mem’**

Example: 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’

4.7.2.6 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 level DEBUG2: “1234.mars;Formula evaluation for job had an error. Zero value will be used".

4.7.2.7 Logging

For each job, the evaluated formula answer is logged at the highest logging level (DEBUG3):

```
"Formula  Evaluation = <answer>"
```

4.7.3 Eligible Wait Time for Jobs

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 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.

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 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.

accrue_type Job attribute. The type of time that the job is accruing. This can be one of `eligible_time`, `ineligible_time`, `run_time`, `exit_time` or `initial_time`.

A job accrues `ineligible_time` while it is blocked by user or group limits, such as:

max_user_run
max_group_run
max_user_res.RES
max_group_res.RES
max_user_run_soft
max_group_run_soft
max_user_res_soft.RES
max_group_res_soft.RES

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 qmoved or moved due to peer scheduling.

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.

4.7.3.1 The Server's eligible_time_enable Attribute

The server's `eligible_time_enable` attribute controls whether a job's `eligible_time` attribute is used as its starving time. See section 4.20 "Starving Jobs" on page 233.

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.

4.7.3.2 The Job's accrue_type Attribute

The job's accrue_type attribute indicates what kind of time the job is accruing.

Table 6: Job's accrue_type Attribute

Type	Numeric Representation	Type
JOB_INITIAL	0	initial_time
JOB_INELIGIBLE	1	ineligible_time
JOB_ELIGIBLE	2	eligible_time
JOB_RUNNING	3	run_time
JOB_EXIT	4	exiting

The job's accrue_type attribute is visible via qstat only by Manager, and is set only by the server.

4.7.3.3 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 DEBUG3 level.

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

Example:
08/07/2007 13:xx:yy;0040;Server@pepsi;Job;163.pepsi;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@pepsi;Job;163.pepsi;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@pepsi;Job;163.pepsi;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@pepsi;Job;163.pepsi;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@pepsi;Job;163.pepsi;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@pepsi;Job;163.pepsi;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@pepsi;Job;163.pepsi;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@pepsi;Job;163.pepsi;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

4.7.3.4 Accounting

Each job's eligible_time attribute is included in the “E” records in the PBS accounting logs.

Example
08/07/2007 19:34:06;E;182.blrm243;user=spb group=spb job-name=STDIN queue=workq ctime=1186494765 qtime=1186494765 etime=1186494765 start=1186494767 exec_host=blrm243/0 exec_vnode=(blrm243: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 Advance and Standing Reservations

An *advance reservation* is a reservation for a set of resources for a specified time. The reservation is only available to a specific user or group of users.

A *standing reservation* is 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.

An *instance* of a standing reservation is also called an *occurrence* of the standing reservation. The *soonest occurrence* of a standing reservation is the occurrence which is currently active, or if none is active, then it is the next occurrence.

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 occurrence. You can only specify *when* the job is eligible to run. See the `qsub(1B)` man page.
- 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 instance 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`.

The time for which a reservation is requested is in the time zone at the submission host.

The user creates 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 instances 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<unique integer>.<server name>

For a standing reservation, this reservation ID refers to the entire series, and has the format:

S<unique integer>.<server name>

The user specifies the resources for a reservation using the same syntax as for a job. Jobs in reservations are placed the same way non-reservation jobs are placed in placement sets.

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.

To query a reservation, use the `pbs_rstat` command. See section 8.9.5 “Viewing the Status of a Reservation” on page 185 of the PBS Professional User’s Guide. To delete an advance reservation, use the `pbs_rdel` command, not the `qmgr` command.

For detailed information on creation and use of reservations, see section 8.9 “Advance and Standing Reservation of Resources” on page 172 of the PBS Professional User’s Guide.

When a reservation is created, it is created within a placement set. A job within a reservation runs within that placement set, but that is the only placement set considered for the job.

A job running in a reservation cannot be preempted.

4.8.1 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.

4.8.2 Advance and Standing Reservations and FLEX Licensing

Reservation jobs won’t run if PBS runs out of FLEX licenses. Set the server’s `pbs_license_min` attribute to the total number of CPUs, including virtual CPUs, in the PBS complex. See section 5.7.1.3 “Licensing and Reservations” on page 107 in the PBS Professional Installation & Upgrade Guide and section 5.4.3 “Setting Server Licensing Attributes” on

page 95 in the PBS Professional Installation & Upgrade Guide.

4.8.3 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.

4.8.4 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 section 2.6 “Server Configuration Attributes” on page 18.

Table 7: Server Attributes Affecting Reservations

Attribute	Effect
acl_resv_host_enable	Controls whether or not the server uses the acl_resv_hosts access control lists.
acl_resv_hosts	List of hosts from which reservations may and may not be created at this server.
acl_resv_group_enable	Controls whether or not the server uses the acl_resv_groups access control lists.
acl_resv_groups	List of groups who may and may not create reservations at this server.
acl_resv_user_enable	Controls whether or not the server uses the acl_resv_users access control lists.
acl_resv_users	List of users who may and may not create reservations at this server.
resv_enable	Controls whether or not reservations can be created at this server.

The vnode attributes that affect reservations are listed here, and described in section 2.9 “Vnode Configuration Attributes” on page 53.

Table 8: Vnode Attributes Affecting Reservations

Attribute	Effect
queue	Associates the vnode with an execution queue. If this attribute is set, this vnode cannot be used for reservations.
resv_enable	Controls whether the vnode can be used for reservations.

4.9 How Queues are Ordered

The order in which jobs are considered by the scheduler depends upon which queues those jobs are in, and the ordering of those queues. A queue’s priority determines where it is in the list of queues examined. If queues don’t have priority assigned to them, then the order in which they are considered is essentially random. So if you wish to have queues considered in a particular order, give each queue a different priority.

4.10 Defining Dedicated Time

The file *PBS_HOME*/`sched_priv/dedicated_time` defines the dedicated times for the Scheduler. During dedicated time, only jobs in the dedicated time queues can be run (see `dedicated_prefix` in section 4.3 “Scheduler Configuration Parameters” on page 162). The format of entries is:

```
# From Date-Time   To Date-Time
# MM/DD/YYYY HH:MM MM/DD/YYYY HH:MM
# For example
04/15/2007 12:00 04/15/2007 15:30
```

In order to use a dedicated time queue, jobs must have a walltime. Jobs that do not have a walltime will never run.

To force the Scheduler to re-read the dedicated time file (needed after modifying the file), restart or reinitialize (HUP) the Scheduler. (For details, see section 6.5 “Starting and Stopping PBS: UNIX and Linux” on page 277 and section 6.6 “Starting and Stopping PBS: Windows XP” on page 294.)

4.11 Defining Primetime and Holidays

Often is it useful to change scheduler policy at predetermined intervals over the course of the work week or day. *Prime* and *nonprime* are times when prime or non-primetime start. To have the Scheduler enforce a distinction between primetime (usually, the normal work day) and non-primetime (usually nights and weekends), as well as enforcing non-primetime scheduling policy during holidays, edit the `PBS_HOME/sched_priv/holidays` file to specify the appropriate values for the begin and end of primetime, and any holidays. The ordering is important. Any line that begins with a “*” or a “#” is considered a comment. The format of the holidays file is:

```
YEAR YYYY  This is the current year.
<day> <prime> <nonprime>
<day> <prime> <nonprime>
```

If there is no `YEAR` line in the holidays file, primetime will be in force at all times. *Day* can be *weekday, monday, tuesday, wednesday, thursday, friday, saturday, or sunday.* 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. Each line must have all three fields. In order to have the equivalent of primetime overnight, swap the definitions of prime and non-prime in the scheduler's configuration file.

Times can either be HHMM with no colons(:) or the word "`all`" or "`none`" to specify that a day is all primetime or non-primetime.

<day of year> <date> <holiday>

PBS Professional 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"). *Date* is the calendar date (e.g. "Jan 1"). *Holiday* is the name of the holiday (e.g. "New Year's Day"). Day names must be lowercase.

```
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
```

Reference copies of the holidays file for years 2008, 2009 and 2010 are provided in `PBS_EXEC/etc/holiday.2008`, `PBS_EXEC/etc/holiday.2009`, and `PBS_EXEC/etc/holiday.2010`. The current year's holiday 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.

If backfill_prime is set to True, the scheduler won't run any jobs which would overlap the boundary between primetime and non-primetime. This assures that jobs restricted to running in either primetime or non-primetime can start as soon as the time boundary happens.

If prime_exempt_anytime_queues is set to True, anytime queues are not controlled by backfill_prime, which means that jobs in an anytime queue will not be prevented from running across a primetime/nonprimetime 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.

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 debug level DEBUG2. 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"

4.12 Configuring SMP Cluster Scheduling

The scheduler schedules SMP clusters in an efficient manner. Instead of scheduling only via load average of hosts, it takes into consideration the resources specified at the server, queue, and vnode level. Furthermore, the Administrator can explicitly select the resources to be considered in scheduling via an option in the Scheduler's configuration file (`resources`). The configuration parameter `smp_cluster_dist` allows you to specify how hosts are selected.

The available choices are `pack` (pack one vnode until full), `round_robin` (put one job on each vnode in turn), or `lowest_load` (put one job on the lowest loaded host). The `smp_cluster_dist` parameter should be used in conjunction with `node_sort_key` to ensure efficient scheduling. (Optionally, you may wish to enable "load balancing" in conjunction with SMP cluster scheduling. For details, see section 4.13 “Enabling Load Balancing” on page 213.)

Important: This feature only applies to single-host jobs where the number of chunks is 1, and *place=pack* has been specified.

Note that on a multi-vnode machine, `smp_cluster_dist` will distribute jobs across vnodes but the jobs will end up clustered on a single host.

To use these features requires two steps: setting resource limits via the Server, and specifying the scheduling options. Resource limits are set using the `resources_available` parameter of vnodes via `qmgr` just like on the server or queues. For example, to set maximum limits on a host called “host1” to 10 CPUs and 2 GB of memory:

Qmgr: **set node host1 resources_available.ncpus=10**
Qmgr: **set node host1 resources_available.mem=2GB**

Important: Note that by default both `resources_available.ncpus` and `resources_available.mem` are set to the physical number reported by MOM on the vnode. Typically, you do not need to set these values, unless you do not want to use the actual values reported by MOM.

Next, the Scheduler options need to be set. For example, to enable SMP cluster Scheduler to use 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
```

Finally, specify the resources to use during scheduling:

```
resources: “ncpus, mem, arch, host”
```

4.13 Enabling Load Balancing

The load balancing scheduling algorithm will balance the computational load of single-vnode jobs (i.e. not multi-vnode jobs) across a complex. The load balancing takes into consideration the load on each host as well as all resources specified in the “resource” list. Load balancing uses the value for “loadave” returned by the operating system. For UNIX/Linux, this is the raw one minute averaged "loadave"; for Windows, there is one choice for “loadave”.

When the loadave is above max_load, that node is marked “busy”. The scheduler won’t place jobs on a node marked “busy”. When the loadave drops below ideal_load, the “busy” mark is removed. Consult your OS documentation to determine values that make sense.

The load average will slowly increase over time and more jobs than you want may be started at first. Over a period of time, the load average will move up to a point where no additional jobs will be started on that node. As jobs terminate the load average will slowly move lower and it will take time before the node is the best choice for new jobs.

To configure load balancing, first enable the option in the Scheduler’s configuration file:

```
load_balancing: True ALL
```

Next, configure SMP scheduling as discussed in the previous section, section 4.12 “Configuring SMP Cluster Scheduling” on page 211.

Next, configure the ideal and maximum desired load in each execution host’s MOM configuration file. (See also the discussion of these two MOM options in section 3.2.2 “Syntax and Contents of Default Configuration File” on page 111.)

```
$ideal_load 30
$max_load   32
```

Last, set each host’s `resources_available.ncpus` to the maximum number of CPUs you wish to allocate on that host.

4.14 Managing Load Levels on Hosts

The “loadave” reported by MOM is the raw one minute averaged "loadave" returned by the operating system. When the loadave is above max_load, that node is marked “busy”. The scheduler won’t place jobs on a node marked “busy”. When the loadave drops below ideal_load, the “busy” mark is removed. Consult your OS documentation to determine values that make sense.

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 ideal_load and max_load in MOM’s configuration file to values that are low enough to allow other processes to use some of the host.

If you want to prevent PBS from placing jobs on an already-overloaded machine, set max_load and ideal_load to the values you want for the host. When the load goes above max_load, no more jobs will be run on that host. This will prevent jobs from being started on a host where rogue processes are taking up all the CPU time.

4.15 Enabling Preemptive Scheduling

PBS provides the ability to preempt currently running jobs in order to run higher priority work. Preemptive scheduling is enabled by setting several parameters in the Scheduler’s configuration file (discussed below, and in “Scheduler Configuration Parameters” on page 162). Jobs in advance or standing reservations are not preemptable. If high priority jobs (as defined by your settings on the preemption parameters) can not run immediately, the Scheduler looks for jobs to preempt, in order to run the higher priority job. A job can be preempted in several ways. The Scheduler can suspend the job (i.e. sending a SIGSTOP signal), checkpoint the job (if supported by the underlying operating system, or if the Administrator configures site-specific checkpointing, as described in section 3.5.2 “Site-Specific Job Checkpoint and Restart” on page 125 in the PBS Professional Installation & Upgrade Guide), or requeue the job (a requeue of the job terminates the job and places it back into the queued state). The Administrator can choose the order of these attempts via the `preempt_order` parameter.

Important: If the Scheduler cannot find enough work to preempt in order to run a given job, it will not preempt any work.

When a job is suspended, its FLEX 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 FLEX 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 section 5.7.1.1 "Licensing and Job States" on page 106 in the PBS Professional Installation & Upgrade Guide.

There are several Scheduler parameters to control preemption. The `preemptive_sched` parameter turns preemptive scheduling on and off. You can set the minimum queue priority needed to identify a queue as an express queue via the `preempt_queue_prio` parameter. The `preempt_prio` parameter provides a means of specifying the order of precedence that preemption should take. The ordering is evaluated from left to right. One special name (`normal_jobs`) is the default (If a job does not fall into any of the specified levels, it will be placed into `normal_jobs`.). If you want normal jobs to preempt other lower priority jobs, put `normal_jobs` before them in the `preempt_prio` list. If two or more levels are desired for one priority setting, the multiple levels may be indicated by putting a '+' between them. A complete listing of the preemption levels is provided in the Scheduler tunable parameters section above. The `preempt_order` parameter can be used to specify the preemption method(s) to be used. If one listed method fails, the next one will be attempted.

Soft run limits can be set or unset via `qmgr`. If unset, the limit will not be applied to the job. However if soft run limits are specified on the Server, either of `queue_softlimits` or `server_softlimits` need to be added to the `preempt_prio` line of the Scheduler's configuration file in order to have soft limits enforced by the Scheduler.

The job sort `preempt_priority` will sort jobs by their preemption priority. Note: It is a good idea to put `preempt_priority` as the primary sort key (i.e. `job_sort_key`) if the `preempt_prio` parameter has been modified. This is especially necessary in cases of when soft limits are used. When you are using soft limits, you want to have jobs that are not over their soft limits have higher priority. This is so that a job over its soft limit will not be run, just to be preempted later in the cycle by a job that is not over its soft limits. To do this, use

job_sort_key:"preempt_priority HIGH"

Note that any queue with a priority 150 (default value) or higher is treated as an express (i.e. high priority) queue.

For example: 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 will be preempted until the job from group B can run.

Below is an example of (part of) the Scheduler's configuration file showing how to enable preemptive scheduling and related parameters. Explanatory comments precede each configuration parameter.

```
# 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”
```

4.15.1 Preemption Ordering by Start Time

PBS has a feature that allows a different ordering of preemption of jobs. The default behavior will order preemption of jobs by most recent start time. If "preempt_sort" is disabled, then the first submitted job will be preempted.

For example, if we have two jobs, job A submitted at 10:00 a.m. and job B submitted at 10:30 a.m., the default behavior will preempt job A, and the alternate behavior will preempt job B.

In PBS_HOME/sched_priv/sched_config, the keyword preempt_sort can be set to “min_time_since_start” to enable this alternate behavior.

4.16 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.

4.16.1 Outline of How Fairshare Works

The owner of a PBS job can be defined for fairshare purposes to be a user, a group, an 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" catch-all. All owners in "unknown" get the same resource allotment.

4.16.2 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.

4.16.3 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 two things: in `PBS_HOME/sched_priv/sched_config`, set the scheduler configuration parameter `fair_share` to true, and uncomment the `unknown_shares` setting so that it is set to `unknown_shares: 10`.

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.

4.16.4 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 queues. 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 9: Using Fairshare Entities

Keyword	Fairshare Entities	Purpose
euser	Username	Individual users are allotted shares of the resource being tracked. Each username may only appear once, regardless of group.
egroup	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.
queues	Queues	Shares are allotted between queues.

4.16.4.1 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.

4.16.4.2 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 `unknown` vertex will have 0 shares if `unknown_shares` is commented out. If `unknown_shares` is not commented out, 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.

4.16.4.3 Format for Describing the Tree

The file describing the fairshare tree contains four columns to describe the vertices in the tree. 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.

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 egroup:euser 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 the -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.

4.16.4.4 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:

First, it gives the root of the tree a tree percentage of 100 percent. It proceeds down the tree, finding the tree percentage first for immediate children of root, then their children, ending with leaves.

For each internal vertex A:
sum the shares of its children;
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.

4.16.5 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`. The default for this resource is `cput`, CPU time. Another resource is the exact string "`ncpus*walltime`" which multiplies the number of cpus used by the walltime in seconds. 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.

Note that if a job ends between two scheduling cycles, its resource usage between the end of the job and the following scheduling cycle 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>`**

4.16.6 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.

4.16.7 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.

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.

4.16.8 Files and Parameters Used in Fairshare

`PBS_HOME/sched_priv/sched_config`

`fair_share` [true/false] Enable or disable fairshare

`fairshare_usage_res`
Resource whose usage is to be tracked; default is cput

`half_life` Decay time period; default is 24 hours

`sync_time` Time between writing all data to disk; default 1 hour

`unknown_shares`
Number of shares for unknown vertex; default 10, 0 if commented out

`fairshare_entity`
The kind of entity which is having fairshare applied to it.
Leaves in the tree are this kind of entity. Default: euser.

`fairshare_enforce_no_shares`
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.

`by_queue` If on, queues cannot be designated as fairshare entities, and fairshare will work queue by queue instead of on all jobs at once.

`PBS_HOME/sched_priv/resource_group`
Contains the description of the fairshare tree.

`PBS_HOME/sched_priv/usage`
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.

4.16.9 Fairshare and Queues

The scheduler configuration parameter `by_queue` in the file `PBS_HOME/sched_priv/sched_config` is set to on by default. When `by_queue` is true, fairshare cycles through queues, not overall jobs. So first fairshare is applied to Queue1, then Queue2, etc. If `by_queue` is true, queues cannot be designated as fairshare entities.

4.16.10 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.

4.16.11 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` at an interval set in the scheduler configuration parameter `sync_time`. The default interval is one hour. To have the scheduler write out usage date prior to being killed, issue a **`kill -HUP`**. Otherwise, any usage data acquired since the last write will be lost.

See the `pbsfs(8B)` manual page for more information on using the `pbsfs` command.

4.16.12 Caveats

Do not use fairshare with the combination of strict_ordering and backfilling.

4.17 Enabling Strict Priority

Not to be confused with fairshare (which considers past usage of each entity in the selection of jobs), the scheduler offers a sorting key called "`fair_share_perc`" (see also section 4.3 "Scheduler Configuration Parameters" on page 162). Selecting this option enables the sorting of jobs based on the priorities specified in the fairshare tree (as defined above in the `resource_group` file). A simple share tree will suffice. Every user's `parent_group` should be `root`. The amount of shares should be their desired priority. `unknown_shares` (in the Scheduler's configuration file) should be set to one. Doing so will cause everyone who is not in the tree to share one share between them, making sure everyone else in the tree will have priority over them. Lastly, `job_sort_key` must be set to "`fair_share_perc HIGH`". This will sort by the fairshare tree which was just set up. For example:

```
usr1  60  root  5
usr2  61  root  15
usr3  62  root  15
usr4  63  root  10
usr5  64  root  25
usr6  65  root  30
```

4.18 Enabling Peer Scheduling

PBS Professional includes a feature to have different PBS complexes automatically run jobs from each other's queues. This provides the facility to dynamically load-balance across multiple, separate PBS complexes. These cooperating PBS complexes are referred to as "Peers". In peer scheduling, PBS server A 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 will move the job to the specified queue on the pulling Server and immediately run the job. A job is pulled only when it can run immediately.

You can set up peer scheduling so that A pulls from B and C, and so that B also pulls from A and C.

4.18.1 Prerequisites for Peer Scheduling

The pulling and furnishing queues must be created before peer scheduling can be configured. See section 2.7.2 "Creating Queues" on page 38 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.

Under Windows, if `single_signon_password_enable` is set to "true" among all peer Servers, then users must have their password cached on each Server. For details see section 2.14.3 "Single Signon and Peer Scheduling" on page 89.

4.18.2 Configuring for Peer Scheduling

To configure your complex for peer scheduling, you must:
- Define a flat user namespace on all complexes
- Map pulling queues to furnishing queues
- Grant manager access to each pulling server
- If possible, make user-to-group mappings be consistent across complexes

These steps are described next.

4.18.2.1 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 user names. 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
```

4.18.2.2 Mapping Pulling Queues to Furnishing Queues

You configure for peer scheduling by mapping a furnishing Peer’s queue to a pulling Peer’s queue. You can map a pulling queue to more than one furnishing queue, or more than one pulling queue to a 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 Complex B’s queue “workq”, as well as 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".

4.18.2.3 Granting Manager Access to Pulling Servers

Each furnishing Peer 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@pulling-Server.domain.com
```

For Windows:

```
Qmgr: set server managers += pbsadmin@*
```

4.18.2.4 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 `egroup` 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 `egroup` 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.

4.18.3 Peer Scheduling and Failover Configuration

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 example, the furnishing servers are ServerB1 and ServerB2, the furnishing queues are both called workq, and the pulling server's queue is pull_queue. Configure complex A's peer_queue setting in `PBS_HOME/sched_priv/sched_config` this way:

peer_queue: "pull_queue workq@ServerB1.example.com"
peer_queue: "pull_queue workq@ServerB2.example.com"

4.18.4 Jobs That Have Been Moved to Another Server

Since the Scheduler maps the remote jobs to its own local queue, any moved jobs are subject to the policies of the queue they are moved into. If remote jobs are to be treated differently from local jobs, this can be done on the queue level. A queue can be created exclusively for remote jobs to allow queue level policy to be set for remote jobs. For example, you can set a priority value for each queue, and enable sorting by priority to ensure that pulled jobs are always lower (or higher!) priority than locally submitted jobs. For example, this means that if the local queue for pulled jobs has lower priority, the pulling complex will only pull a job when there are no higher-priority jobs that can run.

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
```

4.19 Using strict_ordering

With strict_ordering, all jobs on the server are considered as a group. This is different from considering first the jobs in one queue, then the jobs in another queue.

With strict_ordering, (sorting at the server level) if there are two queues, and each queue has one starving job and one lower-priority job, those two starving jobs as a group will go ahead of the two lower-priority jobs.

If the jobs were sorted at the queue level, then the starving job in one queue would go, followed by the lower-priority job in that queue, _then_ the starving job in the other queue would go, followed by the other lower-priority job.

Queue A jobs: StarveA, LowPriA
Queue B jobs: StarveB, LowPriB

Order of jobs when sorting at server level:
StarveA & StarveB (in some order like job submission)
LowPriA & LowPriB (ditto)

Order of jobs when sorting at the queue level:
StarveA
LowPriA
StarveB
LowPriB

4.19.1 Enabling FIFO Scheduling with strict_ordering

True first-in, first-out (FIFO) scheduling means sorting jobs into the order submitted, and then running jobs in that order. Furthermore, it means that when the Scheduler reaches a job in the sorted list that cannot run, then no other jobs will be considered until that job can run. In many situations, this results in an undesirably low level of system utilization. However, some customers have a job-mix or a usage policy for which FIFO scheduling is appropriate. When strict_ordering is used, it orders jobs according to the table in section 4.7 “Job Priorities in PBS Professional” on page 193.

Because true FIFO runs counter to many of the efficiency algorithms in PBS Professional, several options must be set in order to achieve true FIFO scheduling within a given queue. In order to have jobs within individual queues be run in true FIFO order, set the following parameters to the indicated values in the Scheduler’s configuration file:

```
strict_ordering:    True   ALL
round_robin:        False  ALL
job_sort_key:       False  ALL
fairshare           False  ALL
help_starving_jobs  False  ALL
backfill:           False  ALL
```

If you are using a single execution queue, you can have true FIFO scheduling for your jobs. You can give priority to queues and have FIFO on all the jobs in the complex in the order in which the queues are sorted.

4.19.2 Combining `strict_ordering` and Backfilling

Strict ordering can be combined with backfilling. If the next job in the ordering cannot run, jobs can be backfilled around the job that cannot run. Note that this is not precisely FIFO anymore.

4.19.3 Caveats

It is inadvisable to use `strict_ordering` and `backfill` 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.

Using dynamic resources with strict_ordering and backfilling may result in unpredictable scheduling. See “Backfilling Caveats” on page 235.

Using preemption with strict_ordering and backfilling may change which job is being backfilled around.

4.20 Starving Jobs

If the `PBS_HOME/sched_priv/sched_config` parameter called `help_starving_jobs` is set to True, then when a job has been waiting to run for a certain amount of time, it is considered to be starving. The amount of time required for a job to be considered starving is set in the `PBS_HOME/sched_priv/sched_config` parameter called `max_starve`.

The server's `eligible_time_enable` attribute controls whether a job's `eligible_time` attribute is used as its starving time. If `eligible_time_enable` is set to True, then a job's `eligible_time` value is used as its wait time for starving. If `eligible_time_enable` is set to False, then the amount of time the job has been queued is used as its wait time for starving. See section 2.6 "Server Configuration Attributes" on page 18 and section 4.7.3 "Eligible Wait Time for Jobs" on page 200.

Starving jobs are assigned the priority level of *starving*. These jobs have higher priority according to the scheduler's standard sorting order. See section 4.7 "Job Priorities in PBS Professional" on page 193. In addition, the order in which starving jobs can preempt other jobs or be preempted is set via the `preempt_prio` configuration option. See "preempt_prio" on page 170.

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.

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.

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.

The following table lists the parameters and attributes that affect starving.

Table 10: Parameters and Attributes Affecting Starving

Parameter or Attribute	Location	Effect
help_starving_jobs	PBS_HOME/sched_priv/sched_config	Controls whether long-waiting jobs are considered starving. When set to true, jobs can be starving. Default: True all
max_starve	PBS_HOME/sched_priv/sched_config	Amount of wait time for job to be considered starving. Default: 24 hours.
eligible_time_enable	Server attribute	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.

Table 10: Parameters and Attributes Affecting Starving

Parameter or Attribute	Location	Effect
eligible_time	Job attribute	The amount of time a job has been blocked from running due to lack of resources.

4.21 Using Backfilling

Backfilling means fitting smaller jobs around the jobs that the scheduler was going to run anyway. Backfilling is only used around starving jobs and with `strict_ordering`. The scheduler keeps track of which job is due to run next (the "most deserving job") according to the policy that has been set, but in addition, it looks for the next job according to policy where that job is also small enough to fit in the available slot (the "small job"). It runs the small job as long as that won't change the start time of the most deserving job due to run next.

The scheduler recalculates everything at each scheduling cycle, so the most deserving job and the small job may change from one cycle to the next.

When `strict_ordering` is on, the scheduler chooses the next job in the standard order. The scheduler also chooses its small job in the standard order. See section 4.7 "Job Priorities in PBS Professional" on page 193

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.

4.21.0.1 Backfilling Caveats

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 most deserving job. The scheduler must estimate when resources will become available, but it can only query for available resources, 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.

Chapter 5

Customizing PBS Resources

It is possible for the PBS Manager to define new resources within PBS. The primary use of this feature is to add site-specific resources, such as to manage software application licenses. This chapter discusses the steps involved in specifying such new resources to PBS, followed by several detailed examples of use.

Once new resources are defined, jobs may request these new resources and the Scheduler will consider the new resources in the scheduling policy. Using this feature, it is possible to schedule resources where the number or amount available is outside of PBS's control.

5.1 Overview of Custom Resource Types

Custom resources can be static or dynamic. Dynamic custom resources can be defined at the server or host. Static custom resources are defined ahead of time, at the server, queue or vnode. Custom resources are defined to the server, then set on one or more vnodes.

For static custom resources the Server maintains the status of the custom resource, and the Scheduler queries the Server for the resource. Static custom resource values at vnode, queue and server can be established via qmgr, setting resources_available.<custom resource name> = <some value>.

For dynamic server-level custom resources the scheduler uses a script to get resource availability. The `script` needs to report the amount of the resource to the Scheduler via `stdout`, in a single line ending with a newline.

For dynamic host-level custom resources, the Scheduler will send a resource query to each MOM to get the current availability for the resource and use that value for scheduling. If the MOM returns a value it will replace the `resources_available` value reported by the Server. If the MOM returns no value, the value from the Server is kept. If neither specify a value, the Scheduler sets the resource value to 0.

For a dynamic host-level resource, values are established by a MOM directive which defines a script which returns a dynamic value via stdout when executed. For a dynamic server-level custom resource, the value is established by the script defined in the **`server_dyn_res`** line in **`PBS_HOME/sched_priv/sched_config`**.

For information on resources shared across vnodes, see “Vnodes and Shared Resources” on page 64.

5.1.1 Custom Resource Formats

The names of custom numeric resources must be alphanumeric with a leading alphabetic: [a-zA-Z][a-zA-Z0-9_]*. Allowable values for float and long resources are the same as for built-in resources. Custom boolean, time, size, string or string array resources must have the same format as built-in resources. See section 2.10.7 “Resource Types” on page 68.

5.2 How to Use Custom Resources

5.2.1 Choosing Dynamic or Static, Server or Host

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.

5.2.2 Using Custom Resources for Application Licenses

The following table lists application licenses and what kind of custom resource to define for them. For specific instructions on configuring each type of license, see examples of configuring custom resources for application licenses in section 5.7 “Application Licenses” on page 256.

Table 1: Custom Resources for Application Licenses

Floating or Node-locked	Unit Being Licensed	How License is Managed	Level	Resource Type
Floating (site-wide)	Token	External license manager	Server	Dynamic
Floating (site-wide)	Token	PBS	Server	Static
Node-locked	Host	PBS	Host	Static
Node-locked	CPU	PBS	Host	Static
Node-locked	Instance of Application	PBS	Host	Static

5.2.3 Using Custom Resources for Scratch Space

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 5.6 “Scratch Space” on page 255 and section 5.4.1 “Dynamic Host-level Resources” on page 247.

5.2.3.1 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. 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. If it is for a host-level resource, it is placed on the host(s) where it will be used. 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.

5.2.4 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 48. Some hosts have a single vnode and some have multiple vnodes. PBS treats all vnodes alike in most respects. Chunks cannot be split across hosts, but they can be split across vnodes on the same host.

Resources that are defined at the host level are applied to vnodes. If you define a dynamic host-level resource, it will be shared among the vnodes on that host. This sharing is managed by the MOM. If you define a static host-level resource, you can set its value at each vnode, or you can set it on one vnode and make it indirect at other vnodes. See “Vnodes and Shared Resources” on page 64.

5.3 Defining New Custom Resources

To define one or more new resources, the Administrator creates or updates the Server resource definition file, *PBS_HOME*/server_priv/resourcedef. Each line in the file defines a new resource.

Once you have defined the new resource(s), you must restart the Server in order for these changes to take effect (see section 5.3.4 on page 246). When the Server restarts, users will be able to submit jobs requesting the new resource, using the normal syntax to which they are accustomed. See also section 5.6 “Scratch Space” on page 255 and section 5.7 “Application Licenses” on page 256.

5.3.1 The resourcedef File

The format of each line in *PBS_HOME*/server_priv/resourcedef is:

```
RESOURCE_NAME [type=RTYPE] [flag=FLAGS]
```

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.

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.

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.

RTYPE is the type of the resource value, which can be one of the following keywords, or will default to long.

See “Resource Types” on page 68 for a description of each resource type. See “Resource Flags for Consumable or Host-level Resources” on page 69 for a description of how resource flags are used.

5.3.2 Defining and Using a Custom Resource

In order for jobs to use a new custom resource, the resource must be:

Step 1 Defined to the server in the server’s `resourcedef` file

Step 2 Put in the “resources” line in `.PBS_HOME/sched_priv/sched_config`

Step 3 Set either via `qmgr` or by adding it to the correct configuration line

Step 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, and if it’s a server-level resource, it must be added to the `server_dyn_res` line

If the 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 String1 on the server, and set it to “foo”, a job requesting “-l String1=bar” will be accepted.

Depending on the type of resource, the server, scheduler and MOMs must be restarted. For detailed steps, see “Configuring Host-level Custom Resources” on page 247 and “Configuring Server-level Resources” on page 252.

5.3.2.1 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
```

2 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, statichos-
tresource,dynamicserverresource, dynamichos-
tresource, staticqueueresource"
```

3 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.static-
serverresource=1
```

See "The qmgr Command" on page 8.

4 If the resource is dynamic:

a. If it's a host-level resource, add it to the "mom_resources" line in `BS_HOME/sched_priv/sched_config`:

```
mom_resources: dynamichostresource
```

Also add it to the MOM config file

```
PBS_HOME/mom_priv/config:
dynamichostresource !path-to-command
```

b. If it's a server-level resource, add it to the "server_dyn_res" line in `PBS_HOME/sched_priv/sched_config`:

```
server_dyn_res: "dynamicserverresource
!path-to-command"
```

.

Table 2: Adding Custom Resources

Resource Type	Server-level	Queue-level	Host-level
static	Set via `qmgr`	Set via `qmgr`	Set via `qmgr`
dynamic	Add to `server_dyn_res` line in `PBS_HOME/sched_priv/sched_config`	Cannot be used.	Add to MOM config file `PBS_HOME/mom_priv/config` and `mom_resources` line in `PBS_HOME/sched_priv/sched_config`

5.3.2.2 Discussion of Scheduling Custom Resources

The last step in creating a new custom resource is configuring the Scheduler to (a) query your new resource, and (b) include the new resource in each scheduling cycle. Whether you set up server-level or host-level resources, the external site-provided script/program is run once per scheduling cycle. Multiple jobs may be started during a cycle. For any job started that requests the resource, the Scheduler maintains an internal count, initialized when the script is run, and decremented for each job started that required the resource.

To direct the Scheduler to use a new server-level custom resource, add the `server_dyn_res` configuration parameter to the Scheduler *PBS_HOME*`/sched_priv/sched_config` file:

```
      server_dyn_res: "RESOURCE_NAME !path-to-command"
```

where `RESOURCE_NAME` should be the same as used in the Server's *PBS_HOME*`/server_priv/resourcedef` file. (See also section 4.3 "Scheduler Configuration Parameters" on page 162).

To direct the Scheduler to use a new dynamic host-level custom resource, add the `mom_resources` configuration parameter to the Scheduler `sched_config` file:

```
      mom_resources: "RESOURCE_NAME"
```

where `RESOURCE_NAME` should be the same as that in the Server's `resourcedef` file and the MOM's `config` file. (see also section 3.2.2 "Syntax and Contents of Default Configuration File" on page 111).

Next, tell the Scheduler to include the custom resource as a constraint in each scheduling cycle by appending the new resource to the `resources` configuration parameter in the Scheduler `sched_config` file:

```
      resources: "ncpus, mem, arch, RESOURCE_NAME"
```

Examples are provided in section 5.6 "Scratch Space" on page 255 and section 5.7 "Application Licenses" on page 256.

Once you have defined the new resource(s), you must restart/reinitialize the Scheduler in order for these changes to take effect (see section 5.3.4 on page 246).

5.3.3 Getting an Accurate Picture of Available Resources

Because some custom resources are external to PBS, they are not completely under PBS' control. 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.

5.3.4 PBS Restart Steps for Custom Resources

In order to have new custom resources recognized by PBS, the individual PBS components must either be restarted or reinitialized for the changes to take effect. The subsequent sections of this chapter will indicate when this is necessary, and refer to the details of this section for the actual commands to type.

The procedures below apply to the specific circumstances of defining custom resources. For general restart procedures, see section 6.5 "Starting and Stopping PBS: UNIX and Linux" on page 277 and section 6.6 "Starting and Stopping PBS: Windows XP" on page 294.

Server restart procedures are:

On UNIX:

```
qterm -t quick
PBS_EXEC/sbin/pbs_server
```

On Windows:

```
Admin> qterm -t quick
Admin> net start pbs_server
```

MOM restart / reinitialization procedures are:

On UNIX: Use the "`ps`" command to determine the process ID of current instance of PBS MOM, and then terminate MOM via `kill` using the PID returned by `ps`. Note that `ps` arguments vary among UNIX systems, thus "`-ef`" may need to be replaced by "`-aux`". Note that if your custom resource gathering script/ program takes longer than the default ten seconds, you can change the alarm timeout via the `-a alarm` command line start option as discussed in section 6.5.3 “Manually Starting MOM” on page 278. You will typically want to use the -p option when starting MOM:

```
ps –ef | grep pbs_mom
kill -HUP <MOM PID>
PBS_EXEC/sbin/pbs_mom -p
```

On Windows:

```
Admin> net stop pbs_mom
Admin> net start pbs_mom
```

If your custom resource gathering script/program takes longer than the default ten seconds, you can change the alarm timeout via the `-a alarm` command line start option as discussed in section 6.6.1 “Startup Options to PBS Windows Services” on page 295.)

Scheduler restart / reinitialization procedures are:

On UNIX:

```
ps –ef | grep pbs_sched
kill -HUP <Scheduler PID>
PBS_EXEC/sbin/pbs_sched
```

On Windows:

```
Admin> net stop pbs_sched
Admin> net start pbs_sched
```

5.4 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.

5.4.1 Dynamic Host-level Resources

A dynamic resource could be scratch space on the host. The amount of scratch space 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.`

These are the steps for configuring a dynamic host-level resource:

Step 1 Write a script, for example `hostdyn.pl`, that returns the available amount of the resource via stdout, and place it on each host where it will be used. For example, it could be placed in `/usr/local/bin/hostdyn.pl`

Step 2 Configure each MOM to use the script by adding the resource and the path to the script in `PBS_HOME/mom_priv/config`.

```
dynscratch !/usr/local/bin/ \
   hostdyn.pl
```

Step 3 Restart the MOMs. See section 5.3.4 “PBS Restart Steps for Custom Resources” on page 246.

Step 4 Define the resource, for example dynscratch, in the server resource definition file *PBS_HOME/server_priv/resourcedef.*

```
dynscratch type=size flag=h
```

Step 5 Restart the server. See section 5.3.4 “PBS Restart Steps for Custom Resources” on page 246.

Step 6 Add the new resource to the “resources” line in `PBS_HOME/sched_priv/sched_config`.

```
resources: “ncpus, mem , arch, dyn-
scratch”
```

Step 7 Restart the scheduler. See section 5.3.4 "PBS Restart Steps for Custom Resources" on page 246.

Step 8 Add the new resource to the "mom_resources" line in PBS_HOME/sched_priv/sched_config. Create the line if necessary.

```
mom_resources: "dynscratch"
```

To request this resource, the resource request would include
-l select=1:ncpus=N:dynscratch=10MB

See section 5.6.1 "Host-level "scratchspace" Example" on page 255 for a more complete discussion of dynamic host-level resources.

The script must return, via stdout, the amount available in a single line ending with a newline.

5.4.1.1 Discussion of Dynamic Host-level Resources

If the new resource you are adding is a dynamic host-level resource, configure each MOM to answer the resource query requests from the Scheduler.

Each MOM can be instructed in how to respond to a Scheduler resource query by adding a shell escape to the MOM configuration file *PBS_HOME*/mom_priv/config. The shell escape provides a means for MOM to send information to the Scheduler. The format of a shell escape line is:

RESOURCE_NAME !*path-to-command*

The RESOURCE_NAME specified should be the same as the corresponding entry in the Server's *PBS_HOME*/server_priv/resourcedef file. The rest of the line, following the exclamation mark ("!"), is saved to be executed through the services of the system(3) standard library routine. The first line of output from the shell command is returned as the response to the resource query.

On Windows, be sure to place double-quote (" ") marks around the *path-to-command* if it contains any whitespace characters.

Typically, what follows the shell escape (i.e. “!”) is the full path to the script or program that you wish to be executed, in order to determine the status and/or availability of the new resource you have added. Once the shell escape script/program is started, MOM waits for output. The wait is by default ten seconds, but can be changed via the `-a alarm` command line start option. (For details of use, see section 6.5.3 “Manually Starting MOM” on page 278 and section 6.6.1 “Startup Options to PBS Windows Services” on page 295.) If the alarm time passes 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, an error is returned, usually to the scheduler, in the form of “`? 15205`". Another log message is written. The `?` indicates an error condition and the value 15205 is `PBSE_RMSYSTEM`. The user’s job may not run.

In order for the changes to the MOM `config` file to take effect, the `pbs_mom` process must be either restarted or reinitialized (see section 5.3.4 on page 246). For an example of configuring scratch space, see section 5.6.1 “Host-level “scratchspace” Example” on page 255.

5.4.2 Static Host-level Resources

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.

Node-locked application licenses can be per-host, where any number of instances can be running on that host, per-CPU, 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.

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:

For a two-CPU job on a single vnode:

```
-l select=1:ncpus=2:license=1
```

For a multi-vnode job:

```
-l select=2:ncpus=2:license=1
-l place=scatter
```

Users can also use “license=True”, but this way they do not have to change their scripts.

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:

For a two-CPU, two-license job:

-l select=1:ncpus=2:license=2

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 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 vnodes need two CPUs each:

```
-l select=2:ncpus=2:license=1
-l place=scatter
```

The rule of thumb is that the chunks have to be the size of a single host so that one license in the chunk corresponds to one license being taken from the host.

These are the steps for configuring a static host-level resource:

Step 1 Define the resource, for example hostlicense, in the server resource definition file *PBS_HOME/* `server_priv/resourcedef.`

For per-CPU or per-use:

hostlicense type=long flag=nh

For per-host:
`hostlicense type=boolean flag=h`

Step 2 Restart the server. See section 5.3.4 “PBS Restart Steps for Custom Resources” on page 246.

Step 3 Use the qmgr command to set the value of the resource on the host.

`Qmgr: set node Host1 hostli-cense=(number of uses, number of CPUs, or True if boolean)`

Step 4 Add the new resource to the “resources” line in PBS_HOME/sched_priv/sched_config.
`resources: “ncpus, mem , arch, hostlicense”`

Step 5 Restart the scheduler. See section 5.3.4 “PBS Restart Steps for Custom Resources” on page 246.

For examples of configuring each kind of node-locked license, see section 5.7.6 “Per-host Node-locked Licensing Example” on page 264, section 5.7.7 “Per-use Node-locked Licensing Example” on page 267, and section 5.7.8 “Per-CPU Node-locked Licensing Example” on page 269.

5.5 Configuring Server-level Resources

5.5.1 Dynamic Server-level Resources

Dynamic server-level resources are usually used for site-wide externally-managed floating licenses. The availability of licenses is determined by running a script or program specified in the `server_dyn_res` line of `PBS_HOME/sched_priv/sched_config`. The script must return the value via stdout in a single line ending with a newline. 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.

Step 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
```

Step 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`

Step 3 Restart the server. See section 5.3.4 "PBS Restart Steps for Custom Resources" on page 246.

Step 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`.

```
server_dyn_res: "floatlicense \
   !/usr/local/bin/serverdyn.pl"
```

Step 5 Add the new dynamic resource to the "resources" line in `PBS_HOME/sched_priv/sched_config`:

```
resources: "ncpus, mem , arch, \
   floatlicense"
```

Step 6 Restart the scheduler. See section 5.3.4 "PBS Restart Steps for Custom Resources" on page 246.

Step 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,node2
Qmgr: set node
resources_available.CanRun=True
```

To request this resource, the job's resource request would include:

```
-l floatlicense=<number of licenses or tokens
required>
-l select=1:ncpus=N:CanRun=1
```

See section 5.6.1 "Host-level "scratchspace" Example" on page 255 for more discussion of dynamic host-level resources.

5.5.2 Static Server-level Resources

Static server-level resources are used for floating licenses that PBS will manage. PBS keeps track of the number of available licenses instead of querying an external license manager.

These are the steps for configuring a static server-level resource:

Step 1 Define the resource, for example sitelicense, in the server resource definition file *PBS_HOME*/ server_priv/resourcedef.

```
sitelicense type=long flag=q
```

Step 2 Restart the server. See section 5.3.4 "PBS Restart Steps for Custom Resources" on page 246.

Step 3 Use the qmgr command to set the value of the resource on the server.

```
Qmgr: set server sitelicense=(num-
ber of licenses)
```

Step 4 Add the new resource to the "resources" line in PBS_HOME/sched_priv/sched_config.

```
resources: "ncpus, mem , arch,
sitelicense"
```

Step 5 Restart the scheduler. See section 5.3.4 "PBS Restart Steps for Custom Resources" on page 246.

5.6 Scratch Space

5.6.1 Host-level "scratchspace" Example

Say you have jobs that require a large amount of scratch disk space during their execution. To ensure that sufficient space is available when starting the job, you first write a script that returns via stdout a single line (with new-line) the amount of space available. This script is placed in `/usr/local/bin/scratchspace` on each host. Next, edit the Server's resource definition file, (`PBS_HOME/server_priv/resourcedef`) adding a definition for the new resource. (See also "Defining New Resources" on page 78.) For this example, let's call our new resource "scratchspace". We'll set **`flag=h`** so that users can specify a minimum amount in their select statements.

```
scratchspace    type=size flag=h
```

Now restart the Server (see section 5.3.4 on page 246).

Once the Server recognizes the new resources, you may optionally specify any limits on that resource via `qmgr`, such as the maximum amount available of the new resources, or the maximum that a single user can request. For example, at the `qmgr` prompt you could type:

```
set server resources_max.scratchspace=1gb
```

Next, configure MOM to use the `scratchspace` script by entering one line into the `PBS_HOME/mom_priv/config` file:

On UNIX:

```
scratchspace !/usr/local/bin/scratchspace
```

On Windows:

```
scratchspace !”c:\Program Files\PBS
Pro\scratchspace”
```

Then, restart / reinitialize the MOM (see section 5.3.4 on page 246).

Edit the Scheduler configuration file (`PBS_HOME/sched_priv/sched_config`), specifying this new resource that you want queried and used for scheduling:

```
mom_resources:  “scratchspace”
resources: “ncpus, mem, arch, scratchspace”
```

Then, restart / reinitialize the Scheduler (see section 5.3.4 on page 246).

Now users will be able to submit jobs which request this new “scratch-space” resource using the normal `qsub -l` syntax to which they are accustomed.

```
% qsub -l scratchspace=100mb ...
```

The Scheduler will see this new resource, and know that it must query the different MOMs when it is searching for the best vnode on which to run this job.

5.7 Application Licenses

5.7.1 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-CPU, per-use or per-host.

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.

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.

5.7.2 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 *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.

5.7.3 Simple Floating License Example

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 3:

Application	Licenses	Hosts	DESCRIPTION
AppF	4	3-6	uses licenses from an externally managed pool

For the floating licenses, we will use two resources. One is a dynamic server resource for the licenses themselves. The other is a boolean resource used to indicate that the floating license can be used on a given host.

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:

   AppF type=long
   runsAppF type=boolean flag=h

2. Restart the Server (see section 5.3.4 on page 246).

Host Configuration

3. Set the boolean resource on the hosts where the floating licenses can be used.

```
qmgr: active node
host3,host4,host5,host6
qmgr: set node
resources_available.runsAppF = True
```

Scheduler Configuration

Edit the Scheduler configuration file.

```
cd $PBS_HOME/sched_priv/
```
[edit] sched_config

4. Append the new resource names to the "`resources:`" line:

```
resources: "ncpus, mem, arch, host,
AppF, runsAppF"
```

5. 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"
```

6. Restart or reinitialize the Scheduler (see section 5.3.4 on page 246).

To request a floating license for AppF and a host on which AppF can run:

```
qsub -l AppF=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 = 1
host4
    resources_available.runsAppF = 1
host5
    resources_available.runsAppF = 1
host6
    resources_available.runsAppF = 1
```

5.7.4 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.

A common question that arises among PBS Professional customers is regarding how to use the dynamic resources to coordinate external floating license checking for applications. The following example illustrates how to implement such a custom resource. Our example needs four features to run an application, so we need four custom resources.

To continue with the example, there are four features required to execute an application, thus *PBS_HOME*/`server_priv/resourcedef` needs to be modified:

```
feature1   type=long
feature3   type=long
feature6   type=long
feature8   type=long
```

Important: Note that in the above example the optional `FLAG` (third column of the `resourcedef` file) is not shown because it is a server-level resource which is not consumable.

Once these resources have been defined, you will need to restart the PBS Server (see section 5.3.4 on page 246).

Now that PBS is aware of the new custom resources we can begin configuring the Scheduler to query the license manager server, and schedule based on the availability of the licenses.

Within *PBS_HOME*/`sched_priv/sched_config` the following parameters will need to be updated, or introduced depending on your site configuration. The '`resources:`' parameter should already exist with some default PBS resources declared, and therefore you will want to append your new custom resources to this line, as shown below.

```
resources: “ncpus, mem, arch, feature1,
feature3, feature6, feature8”
```

You will also need to add the parameter '`server_dyn_res` which allows the Scheduler to execute a program or script, that will need to be created, to query your license manager server for available licenses. For example.

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]”
```

Once the `PBS_HOME/sched_priv/sched_config` has been updated, you will need to restart/reinitialize the `pbs_sched` process.

Essentially, the provided `script` needs to report the number of available licenses to the Scheduler via an echo to `stdout`. Complexity of the script is entirely site-specific due to the nature of how applications are licensed. For instance, an application may require `N+8` units, where `N` is number of CPUs, to run one job. Thus, the script could perform a conversion so that the user will not need to remember how many units are required to execute an `N` CPU application.

5.7.5 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 only run 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
   ```

2. Restart the Server (see section 5.3.4 on page 246).

Host Configuration

3. Set the value of runsAppM on the hosts. (Ensure that 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

Edit the Scheduler configuration file.

cd $PBS_HOME/sched_priv/
[edit] sched_config

4. Append the new resource name to the "`resources:`" line.

   ```
   resources: "ncpus, mem, arch, host,
   AppM, runsAppM"
   ```

5. Restart or reinitialize the Scheduler (see section 5.3.4 on page 246).

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
```

5.7.6 Per-host Node-locked Licensing Example

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 4:

Application	Licenses	Hosts	DESCRIPTION
AppA	1	1-4	uses a local node-locked application license

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.

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
   ```

2. Restart the Server (see section 5.3.4 on page 246).

Host Configuration

3. Set the value of runsAppA on the hosts. (Ensure that 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

Edit the Scheduler configuration file.

cd $PBS_HOME/sched_priv/
[edit] sched_config

4. Append the new resource name to the "`resources:`" line.

   ```
   resources: "ncpus, mem, arch, host,
   AppA"
   ```

5. Restart or reinitialize the Scheduler (see section 5.3.4 on page 246).

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
```

5.7.7 Per-use Node-locked Licensing Example

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 5:

Application	Licenses	Hosts	DESCRIPTION
AppB	2	1-2	uses a local node-locked application license

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 section 5.3.4 on page 246).

Host Configuration

3. Set the value of AppB on the hosts to the maximum number of instances allowed. (Ensure that 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

Edit the Scheduler configuration file.

```
cd $PBS_HOME/sched_priv/
```
[edit] sched_config

4. Append the new resource name to the "`resources:`" line. Host-level boolean resources do not need to be added to the "resources" line.

```
resources: "ncpus, mem, arch, host,
AppB"
```

5. Restart or reinitialize the Scheduler (see section 5.3.4 on page 246).

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
```

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
    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
```

5.7.8 Per-CPU Node-locked Licensing Example

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 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 6:

Application	Licenses	Hosts	DESCRIPTION
AppC	4	3-4	uses a local node-locked application license

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 section 5.3.4 on page 246).

Host Configuration

3. Set the value of AppC on the hosts. (Ensure that 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

Edit the Scheduler configuration file.

`cd $PBS_HOME/sched_priv/`
[edit] sched_config

4. Append the new resource name to the "`resources:`" line. Host-level boolean resources do not need to be added to the "resources" line.

   UNIX:

   ```
   resources: "ncpus, mem, arch, host,
   AppC"
   ```

   Windows:

   ```
   resources: "ncpus, mem, arch, host,
   AppC"
   ```

5. Restart or reinitialize the Scheduler (see section 5.3.4 on page 246).

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.

```
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
```

5.8 Deleting Custom Resources

If the administrator deletes a resource definition from $PBS_HOME/server_priv/resourcedef and restarts the server, any and all jobs which requested that resource will be purged from the server when it is restarted. Therefore removing any custom resource definition should be done with extreme care.

Chapter 6

Integration & Administration

This chapter covers information on integrations and the maintenance and administration of PBS, and is intended for the PBS Manager. Topics covered include: starting and stopping PBS, security within PBS, prologue/epilogue scripts, accounting, configuration of the PBS GUIs, and using PBS with other products such as Globus.

6.1 New Features

6.1.1 Job-specific Staging and Execution Directories

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 creates 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 the directory specified by MOM's $jobdir_root does not exist, PBS will create the job-specific staging and execution directory under the user's home directory. See section 6.14 "The Job's Staging and Execution Directories" on page 330.

6.2 pbs.conf

During the installation of PBS Professional, the pbs.conf file was created as either
/etc/pbs.conf (UNIX) or [PBS Destination Folder]\pbs.conf (Windows, where [PBS Destination Folder] is the path specified when PBS was installed on the Windows platform, e.g., "C:\Program Files\PBS Pro\pbs.conf".) The installed copy of pbs.conf is similar to the one below.

```
PBS_EXEC=/usr/pbs
PBS_HOME=/var/spool/PBS
PBS_START_SERVER=1
PBS_START_MOM=1
PBS_START_SCHED=1
PBS_SERVER=hostname.domain
```

This configuration file controls which components are to be running on the local system, directory tree location, and various runtime configuration options. Each vnode in a complex should have its own pbs.conf file. The following table describes the available parameters :

Table 1:

Parameters	Meaning
PBS_BATCH_SERVICE_PORT	Port Server listens on
PBS_BATCH_SERVICE_PORT_DIS	DIS Port Server listens on
PBS_SYSLOG	Controls use of syslog facility
PBS_SYSLOGSEVR	Filters syslog messages by severity
PBS_ENVIRONMENT	Location of `pbs_environment` file
PBS_EXEC	Location of PBS bin and sbin directories
PBS_HOME	Location of PBS working directories
PBS_LOCALLOG	Enables logging to local PBS log files
PBS_MANAGER_GLOBUS_SERVICE_PORT	Port Globus MOM listens on
PBS_MANAGER_SERVICE_PORT	Port MOM listens on
PBS_MOM_GLOBUS_SERVICE_PORT	Port Globus MOM listens on
PBS_MOM_HOME	Location of MOM working directories
PBS_MOM_SERVICE_PORT	Port MOM listens on
PBS_PRIMARY	Hostname of primary Server
PBS_RCP	Location of `rcp` command if `rcp` is used

Table 1:

Parameters	Meaning
PBS_SCP	Location of `scp` command if `scp` is used; setting this parameter causes PBS to first try `scp` rather than `rcp` for file transport.
PBS_SCHEDULER_SERVICE_PORT	Port Scheduler listens on
PBS_SECONDARY	Hostname of secondary Server
PBS_SERVER	Hostname of host running the Server
PBS_START_SERVER	Set to 1 if Server is to run on this vnode
PBS_START_MOM	Set to 1 if a MOM is to run on this vnode
PBS_START_SCHED	Set to 1 if Scheduler is to run on this vnode

6.3 Environment Variables

The settings in $PBS_HOME/pbs_environment are available to user job scripts. You have to HUP the MOM if you change the file. This file is useful for setting environment variables for mpirun etc.

6.4 Ports

PBS daemons listen for inbound connections at specific network ports. These ports have defaults, but can be configured if necessary. For the list of default ports and information on configuring ports, see section 4.10 “Network Addresses and Ports” on page 70 in the PBS Professional Installation & Upgrade Guide. PBS daemons use ports numbered less than 1024 for outbound communication. For PBS daemon-to-daemon communication over TCP, the originating daemon will request a privileged port for its end of the communication.

6.5 Starting and Stopping PBS: UNIX and Linux

The daemons of PBS can be started by two different methods. These methods are not equivalent. The first method is to use the PBS start/stop script, and the second is to run the command that starts the daemon. When you run the PBS start/stop script, PBS will create any vnode definition files. These are not created through the method of running the command that starts a daemon.

The Server, Scheduler, MOM and the optional MOM Globus processes must run with the real and effective uid of root. Typically the components are started automatically by the system upon reboot. The location of the boot-time start/stop script for PBS varies by OS, shown in the following table.

Table 2:

OS	Location of PBS Startup Script
AIX	/etc/rc.d/rc2.d/S90pbs
HP-UX	/sbin/init.d/pbs
Linux	/etc/init.d/pbs /etc/rc.d/init.d/pbs (on some older linux versions)
OSF1	/sbin/init.d/pbs
Solaris	/etc/init.d/pbs

The PBS startup script reads the `pbs.conf` file to determine which components should be started.

6.5.1 Creation of Configuration Files

When the MOM on a vnode is started via the PBS start/stop script, PBS creates any PBS reserved MOM configuration files. These are not created by the MOM itself, and will not be created when MOM alone is started. Therefore, if you make changes to the number of CPUs or amount of memory that is available to PBS, or if a non-PBS process releases a cpuset, you should restart PBS in order to re-create the PBS reserved MOM configuration files. See section 3.2 “MOM Configuration Files” on page 109.

The startup script can also be run by hand to get status of the PBS components, and to start/stop PBS on a given host. The command-line syntax for the startup script is:

```
STARTUP_SCRIPT [ status | stop | start |
restart ]
```

Alternatively, you can start the individual PBS components manually, as discussed in the following sections. Furthermore, you may wish to change the start-up options, as discussed below.

Important: The method by which the Server and MOMs are shut down and restarted has different effects on running jobs; review section 6.5.8 “Impact of Shutdown / Restart on Running Jobs” on page 292.

6.5.2 Starting MOM on the Altix

The cpusetted MOM can be directed to use existing CPU and memory allocations for cpusets. See the option “-p” on page 282.

6.5.3 Manually Starting MOM

If you start MOM before the Server, she will be ready to respond to the Server’s “are you there?” ping. However, for a cpusetted Altix, MOM must be started using the PBS startup script.

6.5.3.1 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 a 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> pnames = RESOURCE[,RESOURCE
...]
```

6.5.3.2 Manual Creation of cpusets Not Managed by PBS

You may wish to create cpusets not managed by PBS on an Altix running ProPack 4 or greater. 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.

6.5.3.3 Preserving Existing Jobs When Re-starting MOM

MOM by hand, you may wish to keep long-running jobs in the running state, and tell MOM to track them. When stopping MOM, use the INT signal SIGINT to leave the jobs running. When stopping the server and all MOMs, use `qterm -t quick -m`. When starting MOM, if you use the pbs_mom command with no options, MOM will allow existing jobs to continue to run. Use the -p option to the pbs_mom command to tell MOM to track the jobs. The PBS start script will kill any running jobs when it is used.

If you are running PBS on an Altix running ProPack 4 or 5, note that the -p option will tell MOM to use existing cpusets.

Start MOM with the command line:

***PBS_EXEC*/sbin/pbs_mom -p**

6.5.3.4 Restarting MOM After a Reboot

When a UNIX/Linux operating system is first booted, it begins to assign process IDs (PIDs) to processes as they are created. PID 1 is always assigned to the system "init" process. As new ones are created, they are either assigned the next PID in sequence or the first empty PID found, which depends on the operating system implementation. Generally, the session ID of a session is the PID of the top process in the session.

The PBS MOM keeps track of the session IDs of the jobs. If only MOM is restarted on a system, those session IDs/PIDs have not changed and apply to the correct processes.

If the entire system is rebooted, the assignment of PIDs by the system will start over. Therefore the PID which MOM thinks belongs to an earlier job will now belong to a different later process. If you restart MOM with -p, she will believe the jobs are still valid jobs and the PIDs belong to those jobs. When she kills the processes she believes to belong to one of her earlier jobs, she will now be killing the wrong processes, those created much later but with the same PID as she recorded for that earlier job.

Never restart pbs_mom with the -p or the -r option following a reboot of the host system.

6.5.3.5 Killing Existing Jobs When Re-starting MOM

If you wish to kill any existing processes, use the -r option to pbs_mom.

Start MOM with the command line:

```
PBS_EXEC/sbin/pbs_mom -r
```

6.5.3.6 Options to pbs_mom

These are the options to the pbs_mom command:

`-a alarm_timeout`
: Number of seconds before alarm timeout. Whenever a resource request is processed, an alarm is set for the given amount of time. If the request has not completed before alarm_timeout, the OS generates an alarm signal and sends it to MOM. Default: 10 seconds. Format: integer.

`-C checkpoint_directory`
: Specifies the path of the directory used to hold checkpoint files. Only valid on systems supporting

checkpoint/restart. The default directory is PBS_HOME/spool/checkpoint. Any directory specified with the -C option must be owned by root and accessible (rwx) only by root to protect the security of the checkpoint files. See the -d option. Format: string.

`-c config_file`
: MOM will read this alternate default configuration file instead of the normal default configuration file upon starting. If this is a relative file name it will be relative to PBS_HOME/mom_priv. If the specified file cannot be opened, pbs_mom will abort. See the -d option.

  MOM's normal operation, when the -c option is not given, is to attempt to open the default configuration file "config" in PBS_HOME/mom_priv. If this file is not present, pbs_mom will log the fact and continue.

`-d home_directory`
: Specifies the path of the directory to be used in place of PBS_HOME by pbs_mom. The default directory is $PBS_HOME. Format: string.

  Note that pbs_mom uses the default directory to find PBS reserved and site-defined configuration files. Use of the -d option is incompatible with these configuration files, since MOM will not be able to find them if the -d option is given.

`-L logfile`
: Specifies an absolute path and filename for the log file. The default is a file named for the current date in `PBS_HOME/mom_logs`. See the `-d` option. Format: string.

`-M TCP_port`
: Specifies the number of the TCP port on which MOM will listen for server requests and instructions. Default: 15002. Format: integer port number

`-n nice_val` Specifies the priority for the pbs_mom daemon. Format: integer

Note that any spawned processes will have a nice value of zero. If you want all MOM's spawned processes to have the specified nice value, use the UNIX nice command instead: "`nice -19 pbs_mom`".

`-p` Specifies that when starting, MOM should track any running jobs, and allow them to continue running. Cannot be used with the -r option. MOM's default behavior is to allow these jobs to continue to run, but not to track them. MOM is not the parent of these jobs.

Altix running ProPack 4 or greater
The Altix ProPack 4 cpuset pbs_mom will, if given the -p flag, use the existing CPU and memory allocations for cpusets. The default behavior is to remove these cpusets. Should this fail, MOM will exit, asking to be restarted with the -p flag.

`-r` Specifies that when starting, MOM should kill any job processes, mark the jobs as terminated, and notify the server. Cannot be used with the -p option. MOM's default behavior is to allow these jobs to continue to run. MOM is not the parent of these jobs.

Do not use the -r option after a reboot, because process IDs of new, legitimate tasks may match those MOM was previously tracking. If they match and MOM is started with the -r option, MOM will kill the new tasks.

`-R UDP_port` Specifies the number of the UDP port on which MOM will listen for pings, resource information requests, communication from other MOMs, etc. Default: 15003. Format: integer port number.

`-S server_port`

Specifies the number of the TCP port on which pbs_mom initially contact the server. Default: 15001. Format: integer port number.

`-s script_options`

This option provides an interface that allows the administrator to add, delete, and display MOM's configuration files. See section 3.2 “MOM Configuration Files” on page 109. See the following table for a description of using `script_options`:

Table 3: How -s Option is Used

-s insert <scriptname> <inputfile>	Reads inputfile and inserts its contents in a new site-defined pbs_mom configuration file with the filename scriptname. If a site-defined configuration file with the name scriptname already exists, the operation fails, a diagnostic is presented, and pbs_mom exits with a nonzero status. Scripts whose names begin with the prefix "PBS" are reserved. An attempt to add a script whose name begins with "PBS" will fail. pbs_mom will print a diagnostic message and exit with a nonzero status.
-s remove <scriptname>	The configuration file named scriptname is removed if it exists. If the given name does not exist or if an attempt is made to remove a script with the reserved "PBS" prefix, the operation fails, a diagnostic is presented, and pbs_mom exits with a nonzero status.
-s show <scriptname>	Causes the contents of the named script to be printed to standard output. If scriptname does not exist, the operation fails, a diagnostic is presented, and pbs_mom exits with a nonzero status
-s list	Causes pbs_mom to list the set of PBS reserved and site-defined configuration files in the order in which they will be executed.

`-x` Disables the check for privileged-port connections.

6.5.4 Manually Starting the Server

Normally the PBS Server is started from the system boot file via a line such as:

***PBS_EXEC*/sbin/pbs_server [options]**

The command line options for the Server include:

`-A acctfile` Specifies an absolute pathname of the file to use as the accounting file. If not specified, the file is named for the current date in the `PBS_HOME/server_priv/accounting` directory.

`-a active` Specifies if scheduling is active or not. This sets the Server attribute `scheduling`. If the option argument is "true" ("True", "t", "T", or "1"), the server is *active* and the PBS Scheduler will be called. If the argument is "false" ("False", "f", "F", or "0), the server is *idle*, and the Scheduler will not be called and no jobs will be run. If this option is not specified, the server will retain the prior value of the `scheduling` attribute.

`-C` The server starts up, creates the database, and exits. Windows only.

`-d serverhome`

Specifies the path of the directory which is home to the Server's configuration files, `PBS_HOME`. The default configuration directory is `PBS_HOME` which is defined in `/etc/pbs.conf.`

`-e mask` Specifies a log event mask to be used when logging. See "log_events" on page 24.

`-F seconds` Specifies the delay time (in seconds) from detection of possible Primary Server failure until the Secondary Server takes over.

`-G globus_RPP`

Specifies the port number on which the Server should query the status of PBS MOM Globus process. Default is 15006.

`-g globus_port`

Specifies the host name and/or port number on which the Server should connect the PBS MOM Globus process. The option argument, *globus_port*, has one of the forms: `host_name`, `[:]port_number`, or `host_name:port_number`. If `host_name` not specified, the local host is assumed. If `port_number` is not specified, the default port is assumed. Default is 15005.

`-L logfile` Specifies an absolute pathname of the file to use as the log file. If not specified, the file is one named for the current date in the `PBS_HOME/server_logs directory`; see the `-d` option.

`-M mom_port` Specifies the host name and/or port number on which the server should connect to the MOMs. The option argument, *mom_port*, has one of the forms: `host_name`, `[:]port_number`, or `host_name:port_number`. If `host_name` not specified, the local host is assumed. If `port_number` is not specified, the default port is assumed. See the `-M` option for `pbs_mom`. Default is 15002.

`-N` The server runs in standalone mode, not as a Windows service. Windows only.

`-p port` Specifies the port number on which the Server will listen for batch requests. Default is 15001.

`-R RPPport` Specifies the port number on which the Server should query the status of MOM. See the `-R` option for `pbs_mom`. Default is 15003.

`-S sched_port`

Specifies the port number to which the Server should connect when contacting the Scheduler. The option argument, *sched_port*, is of the same syntax as under the `-M` option. Default is 15004.

`-t type` Specifies the impact on jobs when the Server restarts. The *type* argument can be one of the following four options

Table 4:

Option	Effect Upon Job Running Prior to Server Shutdown
cold	All jobs are purged. Positive confirmation is required before this direction is accepted.
create	The Server will discard any existing queues (including jobs in those queues) and re-initialize the Server configuration to the default values. In addition, the Server is idled (scheduling set false). Positive confirmation is required before this direction is accepted.
hot	All jobs in the Running state are retained in that state. Any job that was requeued into the Queued state from the Running state when the server last shut down will be run immediately, assuming the required resources are available. This returns the server to the same state as when it went down. After those jobs are restarted, then normal scheduling takes place for all remaining queued jobs. All other jobs are retained in their current state. If a job cannot be restarted immediately because of a missing resource, such as a vnode being down, the server will attempt to restart it periodically for up to 5 minutes. After that period, the server will revert to a normal state, as if warm started, and will no longer attempt to restart any remaining jobs which were running prior to the shutdown.

Table 4:

Option	Effect Upon Job Running Prior to Server Shutdown
warm	All jobs in the Running state are retained in that state. All other jobs are maintained in their current state. The Scheduler will typically make new selections for which jobs are placed into execution. Warm is the default if `-t` is not specified.

6.5.5 Manually Starting the Scheduler

The Scheduler should also be started at boot time. If starting by hand, use the following command line:

***PBS_EXEC*/sbin/pbs_sched [options]**

There are no required options for the scheduler. Available options are listed below.

`-a alarm` Time in seconds to wait for a scheduling cycle to finish. If this takes too long to finish, an alarm signal is sent, and the scheduler is restarted. If a core file does not exist in the current directory, abort() is called and a core file is generated. The default for **`alarm`** is 1000 seconds.

`assign_ssinodes`
Deprecated. Do not use.

`-d home` This specifies the PBS home directory, `PBS_HOME`. The current working directory of the Scheduler is `PBS_HOME/sched_priv`. If this option is not given, `PBS_HOME` defaults to `PBS_HOME` as defined in the `pbs.conf` file.

`-L logfile` The absolute path and filename of the log file. If this option is not given, the scheduler will open a file named for the current date in the PBS_HOME/sched_logs directory. See the -d option.

`-n` This will tell the scheduler to not restart itself if it receives a sigsegv or a sigbus. The scheduler will by default restart itself if it receives either of these two signals. The scheduler will not restart itself if it receives either one within five minutes of its start.

`-p file` Any output which is written to standard out or standard error will be written to this file. The pathname can be absolute or relative, in which case it will be relative to PBS_HOME/sched_priv. If this option is not given, the file used will be PBS_HOME/sched_priv/sched_out. See the -d option.

`-R port` The port for MOM to use. If this option is not given, the port number is taken from PBS_MANAGER_SERVICE_PORT, in pbs.conf. Default: 15003.

`-S port` The port for the scheduler to use. If this option is not given, the default port number for the PBS scheduler is taken from PBS_SCHEDULER_SERVICE_PORT, in pbs.conf. Default: 15004.

`-N` Instructs the scheduler not to detach itself from the current session.

`--version` The pbs_sched command returns its PBS version information and exits. this option can only be used alone.

The options that specify file names may be absolute or relative. If they are relative, their root directory will be `PBS_HOME/sched_priv`.

6.5.6 Manually Starting Globus MOM

The optional Globus MOM should be started at boot time if Globus support is desired. Note that the provided PBS startup script does not start the Globus MOM. There are no required options. If starting manually, run it with the line:

***PBS_EXEC*/sbin/pbs_mom_globus [options]**

If Globus MOM is taken down and the host system continues to run, the Globus MOM should be restarted with the `-r` option. This directs Globus MOM to kill off processes running on behalf of a Globus job. See the **PBS Professional External Reference Specification** (or the `pbs_mom_globus`(1B) manual page) for a more complete explanation.

If the `pbs_mom_globus` process is restarted without the `-r` option, the assumption that will be made is that jobs have become disconnected from the Globus gatekeeper due to a system restart (cold start). Consequently, `pbs_mom_globus` will request that any Globus jobs that were being tracked and which where running be canceled and requeued.

6.5.7 Stopping PBS

There are two ways to stop PBS. The first is to use the PBS start/stop script, and the second is to use the qterm command.

When you use the pbs start/stop script, by typing “pbs stop”,
the server gets a a qterm -t quick (preserving jobs)
MOM gets a SIGTERM - MOM terminates all running children and exits.

The `qterm` command is used to shut down, selectively or inclusively, the various PBS components. It does not perform any of the other cleanup operations that are performed by the PBS shutdown script. The command usage is:

qterm [-f | -i | -F] [-m] [-s] [-t type] [server...]

The available options, and description of each, follows.

Table 5: qterm Options

(no option)	The `qterm` command defaults to `-t quick` if no options are given.

Table 5: qterm Options

-f	Specifies that the Secondary Server, in a Server failover configuration, should be shut down as well as the Primary Server. If this option is not used in a failover configuration, the Secondary Server will become active when the Primary Server exits. The `-f` and `-i` options cannot be used together
-F	Specifies that the Secondary Server (only) should be shut down. The Primary Server will remain active. The `-F` and `-i` or `-f` options cannot be used together.
-i	Specifies that the Secondary Server, in a Server failover configuration, should return to an idle state and wait for the Primary Server to be restarted. The `-i` and `-f` options cannot be used together.
-m	Specifies that all known `pbs_mom` components should also be told to shut down. This request is relayed by the Server to each MOM. Jobs are left running subject to other options to `qterm`.
-s	Specifies that the Scheduler, `pbs_sched`, should also be terminated.

Table 5: qterm Options

-t <type>	imme-diate	All running jobs are to immediately stop execution. If checkpoint is supported, running jobs that can be checkpointed are checkpointed, terminated, and requeued. If checkpoint is not supported or the job cannot be checkpointed, running jobs are requeued if the rerunnable attribute is true. Otherwise, jobs are killed. Normally the Server will not shut down until there are no jobs in the running state. If the Server is unable to contact the MOM of a running job, the job is still listed as running. The Server may be forced down by a second "qterm `-t immediate`" command.
	delay	If checkpoint is supported, running jobs that can be checkpointed are checkpointed, terminated, and requeued. If a job cannot be checkpointed, but can be rerun, the job is terminated and requeued. Otherwise, running jobs are allowed to continue to run. Note, the operator or Administrator may use the `qrerun` and `qdel` commands to remove running jobs.
	quick	This is the default action if the `-t` option is not specified. This option is used when you wish that running jobs be left running when the Server shuts down. The Server will cleanly shut down and can be restarted when desired. Upon restart of the Server, jobs that continue to run are shown as running; jobs that terminated during the Server's absence will be placed into the exiting state.

If you are not running in Server Failover mode, then the following command will shut down the entire PBS complex:

qterm -s -m

However, if Server Failover is enabled, the above command will result in the Secondary Server becoming active after the Primary has shut down. Therefore, in a Server Failover configuration, the "`-f`" (or the "`-i`") option should be added:

`qterm -s -m -f`

Note that `qterm` defaults to `qterm -t quick.` Also, note that the Server does a quick shutdown upon receiving `SIGTERM`.

Important: Should you ever have the need to stop a single MOM but leave jobs managed by her running, you have two options. The first is to send MOM a SIGINT. This will cause her to shut down in an orderly fashion. The second is to kill MOM with a `SIGKILL` (`-9`). Note that MOM will need to be restarted with the `-p` option in order reattach to the jobs.

6.5.8 Impact of Shutdown / Restart on Running Jobs

The method of how PBS is shut down (and which components are stopped) will affect running jobs differently. The impact of a shutdown (and subsequent restart) on running jobs depends on three things:

1. How the Server (`pbs_server`) is shut down,
2. How MOM (`pbs_mom`) is shut down,
3. How MOM is restarted.

Choose one of the following recommended sequences, based on the desired impact on jobs, to stop and restart PBS:

1. To allow running jobs to continue to run:

Shutdown: **`qterm -t quick -m -s`**

Restart: **`pbs_server -t warm`**
`pbs_mom -p`

pbs_sched

2. To checkpoint and requeue checkpointable jobs, you requeue rerunnable jobs, kill any non-rerunnable jobs, then restart and run jobs that were previously running:

Shutdown: **qterm -t immediate -m -s**

Restart:
```
pbs_mom
pbs_server -t hot
pbs_sched
```

3. To checkpoint and requeue checkpointable jobs, you requeue rerunnable jobs, kill any non-rerunnable jobs, then restart and run jobs without taking prior state into account:

Shutdown: **qterm -t immediate -m -s**

Restart:
```
pbs_mom
pbs_server -t warm
pbs_sched
```

6.5.9 Stopping / Restarting a Single MOM

If you wish to shut down and restart a single MOM, be aware of the following effects on jobs.

Methods of manual shutdown of a single MOM:

Table 6: Methods for Shutting Down a Single MOM

SIGTERM	If a MOM is killed with the signal `SIGTERM`, jobs are killed before MOM exits. Notification of the terminated jobs is not sent to the Server until the MOM is restarted. Jobs will still appear to be in the "R" (running) state.
SIGINT SIGKILL	If a MOM is killed with either of these signals, jobs are not killed before the MOM exits. With `SIGINT`, MOM exits after cleanly closing network connections.

A MOM may be restarted with the following options:

Table 7: MOM Restart Options

pbs_mom	Job processes will continue to run, but the jobs themselves are requeued.
pbs_mom -r	Processes associated with the job are killed. Running jobs are returned to the Server to be requeued or deleted. This option should not be used if the system has just been rebooted as the process numbers will be incorrect and a process not related to the job would be killed.
pbs_mom -p	Jobs which were running when MOM terminated remain running.

6.6 Starting and Stopping PBS: Windows XP

When PBS Professional is installed on either Microsoft Windows XP, the PBS processes are registered as system services. As such, they will be automatically started and stopped when the system boots and shuts down. However, there may come a time when you need to manually stop or restart the PBS services (such as shutting them down prior to a PBS software upgrade). The following example illustrates how to manually stop and restart the PBS services. These lines must be typed at a Command Prompt with Administrator privilege.

```
net stop pbs_sched
net stop pbs_mom
net stop pbs_server
net stop pbs_rshd
```

and to restart PBS:

```
net start pbs_server
net start pbs_mom
net start pbs_sched
net start pbs_rshd
```

It is possible to run (Administrator privilege) the PBS services manually, in standalone mode and not as a Windows service, as follows:

```
Admin> pbs_server -N <options>
Admin> pbs_mom -N <options>
Admin> pbs_sched -N <options>
Admin> pbs_rshd -N <options>
```

6.6.1 Startup Options to PBS Windows Services

The procedure to specify startup options to the PBS Windows Services is as follows:

1 Go to `Start Menu->Control Panel->Performance and Maintenance->AdministrativeTools->Services` (in Windows XP).

2 Select the PBS Service you wish to alter. For example, if you select "PBS_MOM", the MOM service dialog box will come up.

3 Enter in the "Start parameters" entry line as required. For example, to specify an alternate MOM configuration file, you might specify the following input:

```
-c "\Program Files\PBS Pro\home\mom_priv\config2"
```

4 Lastly, click on "Start" to start the specified Service.

Keep in mind that the Windows services dialog does not remember the "Start parameters" value when you close the dialog. For future restarts, you need to always specify the "Start parameters" value.

The pbs_server service has two Windows-specific options. These are:

-C The Server starts up, creates the database, and exits.

-N The Server runs in standalone mode, not as a Windows service.

6.7 Checkpoint / Restart Under PBS

PBS Professional supports two methods of checkpoint/restart: native to the OS, and a generic site-specific method. Operating system checkpoint-restart is supported where provided by the system. Alternatively, a site may configure the generic checkpointing feature of PBS Professional to use checkpoint and restart. For details see section 3.5.2 “Site-Specific Job Checkpoint and Restart” on page 125. In addition, users may manage their own checkpointing from within their application. This is discussed further in the **PBS Professional User’s Guide**.

There are two types of checkpoints:

- The first and most common is what is called *checkpoint and abort*, meaning the job is checkpointed (a restart file is written) and the job is killed, then requeued in the held state (H). This is performed when the qhold is used on a running job, and when the Scheduler preempts a job. The job resumes from the restart file when it is placed into execution again (i.e. run).
- The second and less common is the *snapshot,* where a restart file is written but the job continues to execute. The only time that the job will be resumed from this restart file is when the system crashes.

The location of the directory into which jobs are checkpointed can be specified in a number of ways. In order of preference:

1. “`-C path`” command line option to `pbs_mom`
2. **`PBS_CHECKPOINT_PATH`** environment variable
3. “`$checkpoint_path path`” option in MOM’s config file4
4. default value

Note: checkpointing is not supported for job arrays. On systems that support checkpointing, subjobs are not checkpointed; instead they run to completion.

6.7.1 Manually Checkpointing a Job

On systems which provide OS-level checkpointing, the PBS Administrator may manually force a running job to be checkpointed. This is done by using the `qhold` command. (Discussed in detail in the **PBS Professional Users Guide**). Either:

- The job is checkpointable: The job is checkpointed, then it is killed and requeued. It is marked with hold flag, and is put into the H (held) state.
- The job is not checkpointable: The job keeps running, and is marked with hold flag. The job is put into state R (running).

The `qrls` command is used to release the hold placed through `qhold`. This does not start the job; the job is started when the scheduler selects the job and runs it.

6.7.2 Periodic Checkpointing of a Job

A job can be periodically checkpointed while it is running. The `-c` *interval* option to the `qsub` command sets the interval and the job's `checkpoint` attribute. When this attribute is set, at every interval the job is checkpointed and a restart file is written, but the job keeps running.

6.7.3 Checkpointing Jobs During PBS Shutdown

The PBS start/stop script will not result in PBS checkpointing jobs (on systems which provide OS-level checkpointing). This behavior allows for a faster shutdown of the batch system at the expense of rerunning jobs from the beginning. If you prefer jobs to be checkpointed, then use the `qterm` command, and append the `-t immediate` option.

6.7.4 Suspending/Checkpointing Multi-vnode Jobs

The PBS suspend/resume and checkpoint/restart capabilities are supported for multi-vnode jobs. With checkpoint, the application or OS must be able to save the complete session state in a file. This means any open socket will cause the checkpoint operation to fail. PBS normally sets up a socket connection to a process (`pbs_demux`) which collects stdio streams from all tasks. If this is not turned off, the checkpoint cannot work. Therefore, a job attribute controls this: `no_stdio_sockets`. See the pbs_job_attributes(7B) manual page for more details. If this attribute is true, the pbs_demux process will not be started and no open socket will prevent the checkpoint from working. The other place where PBS will use a socket that must be addressed is if the program pbsdsh is used to spawn tasks. There is a new option for pbsdsh '-o' that is used to prevent it from waiting for the spawned tasks to finish. This is done so no socket will be left open to the MOM to receive task manager events. If this is used, the shell must use some other method to wait for the tasks to finish.

6.8 Security

There are three parts to security in the PBS system:

Internal security	Can the component itself be trusted?
Authentication	How do we believe a client about who it is?
Authorization	Is the client entitled to have the requested action performed?

6.8.1 Internal Security

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**, and **LC_TIME**. The environment file may be edited to include the other variables required on your system.

Important: 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**.

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.

6.8.2 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.

6.8.3 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 the man page for `pbs_mom`(8B) for more information on the configuration file. Access to `pbs_sched` is not limited other than it must be from a privileged port.

6.8.4 User Authentication

The PBS Server authenticates the user name included in a request using the supplied PBS credential. This credential is supplied by `pbs_iff`.

6.8.5 User Authorization

PBS as shipped does not assume a consistent user name space within the set of systems which make up a PBS complex. However, the Administrator can enable this assumption, if desired, by setting the server's flatuid attribute to true. This works when running PBS in an environment that *does* have a flat user namespace. To set the `flatuid` Server attribute to `True` via `qmgr`:

```
Qmgr: set server flatuid=True
```

If `flatuid` is set to `true`, a UserA on HostX who submits a job to the PBS Server on HostY will *not* require an entry in the `/etc/passwd` file (UNIX) or the User Database (Windows), nor a `.rhosts` entry on HostY for HostX, nor must HostX appear in HostY's `hosts.equiv` file. In either case, if a job is submitted by *UserA@HostA,* PBS will allow the job to be deleted or altered by *UserA@HostB*. Note that flatuid may open a security hole in the case where a host has been logged into by someone impersonating a genuine user.

If `flatuid` is *not* set to `true`, a user may supply a name under which the job is to be executed on a certain system (via the `-u user_list` option of the `qsub`(1B) command). If one is not supplied, the name of the job owner is chosen to be the execution name. Authorization to execute the job under the chosen name is granted under the following conditions:

1. The job was submitted on the Server's (local) host and the submitter's name is the same as the selected execution name.

2. The host from which the job was submitted is declared trusted by the execution host in the system `hosts.equiv` file or the submitting host and submitting user's name are listed in the execution users' `.rhosts` file. The system-supplied library function, `ruserok`*()*, is used to make these checks.

The `hosts.equiv` file is located in `/etc` under UNIX, and in `%WINDIR%\system32\drivers\etc\` under Windows).

Additional information on user authorization is given in section 3.4 "UNIX User Authorization" on page 19 in the PBS Professional Installation & Upgrade Guide and section 3.6 "Windows User Authorization" on page 31 in the PBS Professional Installation & Upgrade Guide, as well as in the

PBS Professional User's Guide.

In addition to the above checks, access to a PBS Server and queues within that Server may be controlled by access control lists. (For details see section 2.6 "Server Configuration Attributes" on page 18 and section 2.7.3 "Queue Configuration Attributes" on page 39.)

6.8.6 Group Authorization

PBS allows a user to submit jobs and specify under which group the job should be run at the execution host(s). The user specifies a `group_list` attribute for the job which contains a list of group@host similar to the user list. See the `group_list` attribute under the -W option of `qsub`(1B). The PBS Server will ensure the user is a member of the specified group by:

1. Checking if the specified group is the user's primary group in the password entry on the execution host. In this case the user's name does not have to appear in the group entry for his primary group.

2. Checking on the execution host for the user's name in the specified group entry in `/etc/group` (under UNIX) or in the group membership field of the user's account profile (under Windows).

The job will be aborted if both checks fail. The checks are skipped if the user does not supply a `group_list` attribute (and the user's default/primary group will be used).

Under UNIX, when staging files in or out, PBS also uses the selected execution group for the copy operation. This provides normal UNIX access security to the files. Since all group information is passed as a string of characters, PBS cannot determine if a numeric string is intended to be a group name or GID. Therefore when a group list is specified by the user, PBS places one requirement on the groups within a system: each and every group in which a user might execute a job MUST have a group name and an entry in `/etc/group`. If no `group_list` is used, PBS will use the login group and will accept it even if the group is not listed in `/etc/group`. Note, in this latter case, the `egroup` attribute value is a numeric string representing the GID rather than the group "name".

6.8.7 External Security

In addition to the security measures discussed above, PBS provides three levels of privilege: user, Operator, and Manager. Users have user privilege which allows them to manipulate their own jobs. Manager or Operator privilege is required to set or unset attributes of the Server, queues, vnodes, and to act on other people's jobs. For specific limitations on "user" privilege, and additional attributes available to Managers and Operators, review the following: "section 2.2 "The qmgr Command" on page 8; the introduction to "Administrator Commands" on page 365; and the discussion of user commands in the **PBS Professional User's Guide**.

6.8.8 Enabling Hostbased Authentication on Linux

Hostbased 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.

6.8.8.1 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:

```
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
```

6.8.8.2 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:

```
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 hostbased authentication. These options may be commented out, and so must be uncommented 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/shosts.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 shosts.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.

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.

OpenSSH_3.5p1:
Procedure above should work.

OpenSSH_3.6.1p2:
Procedure above should work with the following additional step:
1. Define "EnableSSHKeysign yes" in the `/etc/ssh/ssh_config` file

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. **`chmod 4755 /usr/lib/ssh/ssh-keysign`**
Was 0755 before chmod.
This file is required to be setuid to work.

NOTE for 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.

6.8.9 Security Considerations for Copying Files

If using Secure Copy (scp), then PBS will first try to deliver output or stagein/out files using scp. If scp fails, PBS will try again using rcp (assuming that scp might not exist on the remote host). If rcp also fails, the above cycle will be repeated after a delay, in case the problem is caused by a temporary network problem. All failures are logged in MOM's log, and an email containing the errors is sent to the job owner.

Attempts:

1a	scp
1b	rcp
2a	scp
2b	rcp
3a	scp
3b	rcp
4a	scp
4b	rcp

6.9 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.

Important: 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.

6.10 Managing PBS and Multi-vnode Parallel Jobs

Many customers use PBS Professional in cluster configurations for the purpose of managing multi-vnode parallel applications. This section provides the PBS Administrator with information specific to this situation.

6.10.1 The PBS_NODEFILE

For each job, PBS will create a job-specific “host file” or “node file”—a text file containing the name of the vnode(s) allocated to that job, listed one per line. The file will be created by the MOM on the first vnode in `PBS_HOME/aux/`*`JOB_ID`*, where *`JOB_ID`* is the actual job identifier for that job. The full path and name for this file is written to the job’s environment via the variable `PBS_NODEFILE`. (See also details on using this environment variable in Chapter 10 of the **PBS Professional User’s Guide**.)

The order in which hosts appear in the PBS_NODEFILE is the order in which chunks are specified in the selection directive. The order in which hostnames appear in the file is hostA X times, hostB Y times, where X is the number of MPI processes on hostA, Y is the number of MPI processes on hostB, etc. See the definition of the resources “mpiprocs” and “ompthreads” in “Resource Types” on page 68.

The number of MPI processes for a job is controlled by the value of the resource mpiprocs. The mpiprocs resource controls the contents of the PBS_NODEFILE on the host which executes the top PBS task for the PBS job (the one executing the PBS job script.) See “Built-in Resources” on page 38. The PBS_NODEFILE contains one line per MPI process with the name of the host on which that process should execute. The number of lines in PBS_NODEFILE is equal to the sum of the values of mpiprocs over all chunks requested by the job. For each chunk with mpiprocs=P, (where P > 0), the host name (the value of the allocated vnode's resources_available.host) is written to the PBS_NODEFILE exactly P times.

The number of OpenMP threads for a job is controlled by the value of the resource ompthreads. The ompthreads resource controls the values of the NCPUS and OMP_NUM_THREADS environment variables for every PBS task (including the top PBS task).

If a chunk requests ncpus=N, with N > 1, PBS will only create one MPI process for that chunk, but set the number of OpenMP threads to N.

6.11 Support for MPI

PBS Professional is tightly integrated with several implementations of MPI. PBS can track resource usage for all of the tasks run under these MPIs. Some of the MPI integrations use pbs_attach, which means MOM polls for usage information like CPU time. The amount of usage data lost between polling cycles will depend on the length of the polling cycle. See “Configuring MOM’s Polling Cycle” on page 122.

6.11.1 Interfacing MPICH with PBS Professional on UNIX

The existing mpirun command can be modified to check for the PBS environment and use the PBS-supplied host file. Do this by editing the `.../mpich/bin/mpirun.args` file and adding the following near line 40 (depending on the version being used):

```
if [ "$PBS_NODEFILE" != "" ]
then
    machineFile=$PBS_NODEFILE
fi
```

Important: Additional information regarding checkpointing of parallel jobs is given in “Suspending/Checkpointing Multi-vnode Jobs” on page 217.

6.11.1.1 MPICH on Linux

On Linux systems running MPICH with P4, the existing `mpirun` command is replaced with `pbs_mpirun` The `pbs_mpirun` command is a shell script which attaches a user’s MPI tasks to the PBS job.

6.11.1.2 The `pbs_mpirun` Command

The PBS command `pbs_mpirun` replaces the standard `mpirun` command in a PBS MPICH job using P4. The usage is the same as `mpirun` except for the -`machinefile` option. The value for this option is generated by `pbs_mpirun`. All other options are passed directly to `mpirun`. The value used for the `-machinefile` option is a temporary file created from the PBS_NODEFILE in the format expected by `mpirun`. If the `-machinefile` option is specified on the command line, a warning will be output saying "Warning, -machinefile value replaced by PBS". The default value for the `-np` option is the number of entries in `PBS_NODEFILE`.

6.11.1.3 Transparency to the User

Users should be able to continue to run existing scripts. To be transparent to the user, `pbs_mpirun` should replace standard `mpirun`. To do this, the link for `mpirun` should be changed to point to `pbs_mpirun`:

- Install MPICH into /usr/local/mpich (or note path for mpirun)

  `mv /usr/local/mpich/bin/mpirun /usr/local/mpich/bin/mpirun.std`
- Create link called “mpirun” pointing to pbs_mpirun in /usr/local/mpich/bin/
- Edit pbs_mpirun to change "mpirun" call to "mpirun.std"

At this point, using "mpirun" will actually invoke pbs_mpirun.

When `pbs_mpirun` is run, it runs `pbs_attach`, which attaches the user’s MPI process to the job.

6.11.1.4 Environment Variables and PATHs

The PBS_RSHCOMMAND environment variable should not be set by the user. For `pbs_mpirun` to function correctly for users who require the use of `ssh` instead of `rsh`, several approaches are possible:

Set P4_RSHCOMMAND in the login environment.

Set P4_RSHCOMMAND externally to the login environment, then pass the value to PBS via qsub(1)'s -v or -V arguments:

```
qsub -vP4_RSHCOMMAND=ssh ...
```

or

```
qsub -V ...
```

A PBS administrator may set P4_RSHCOMMAND in the pbs_environment file in PBS_HOME and advise users to not set P4_RSHCOMMAND in the login environment

PATH on remote machines must contain PBS_EXEC/bin. Remote machines must all have pbs_attach in the PATH.

6.11.1.5 Notes

When using SuSE Linux, use "`ssh -n`" in place of "`ssh`".

Usernames must be identical across vnodes.

6.11.2 Integration with LAM MPI

6.11.2.1 The pbs_lamboot Command

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's 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` program will not redundantly consult the $PBS_NODEFILE if it has been instructed to boot the hosts using the `tm` module. This instruction happens when an argument is passed to `pbs_lamboot` containing "-ssi boot tm" or when the `LAM_MPI_SSI_boot` environment variable exists with the value `tm`.

6.11.2.2 The pbs_mpilam Command

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.

6.11.2.3 PATH

The PATH for `pbs_lamboot` and `pbs_mpilam` on all remote machines must contain PBS_EXEC/bin.

6.11.2.4 Transparency to the User

Both pbs_lamboot and pbs_mpilam should be transparent to the user. Users should be able to run existing scripts.

To be transparent to the user, pbs_lamboot should replace LAM lamboot. The link for lamboot should be changed to point to pbs_lamboot.

1 Install LAM MPI into `/usr/local/lam-<version>`

```
mv /usr/local/lam-<version>/bin/lamboot
/user/local/lam-<version>/bin/lamboot.lam
```

2 Edit `pbs_lamboot` to change "lamboot" call to "lamboot.lam"

3 Rename `pbs_lamboot` to `lamboot`:

```
cd /usr/local/lam-<version>/bin
ln  -s PBS_EXEC/bin/pbs_lamboot lamboot
```

At this point, using "lamboot" will actually invoke `pbs_lamboot`.

To be transparent to the user, `pbs_mpilam` should replace `LAM mpirun`. The link for `mpirun` should be changed to point to `pbs_mpilam`.

1 Install LAM MPI into `/usr/local/lam-<version>`

```
mv /usr/local/lam-<version>/bin/mpirun
/user/local/lam-<version>/bin/mpirun.lam
```

2 Edit `pbs_mpirun` to change “mpirun” call to “mpirun.lam”

3 Rename `pbs_mpilam` to `mpirun`:

```
cd /usr/local/lam-<version>/bin
ln -s PBS_EXEC/bin/pbs_mpilam mpirun
```

Either LAMRSH or LAM_SSI_rsh_agent will need to have the value "ssh -x", depending on whether you are using rsh or ssh.

6.11.3 Integration with HP MPI on HP-UX and Linux

6.11.3.1 The pbs_mpihp Command

The PBS command pbs_mpihp replaces the standard mpirun and mpiexec commands in a PBS HP MPI job on HP-UX and Linux, 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.

6.11.3.2 Transparency to the User

To be transparent to the user, `pbs_mpihp` should replace HP `mpirun`. The recommended steps for making `pbs_mpihp` transparent to the user are:

1 Rename HP’s `mpirun`:

```
cd <MPI installation location>/bin
mv mpirun mpirun.hp
```

2 Link the user-callable “mpirun” to `pbs_mpihp`:

```
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`:

```
cd $PBS_EXEC/etc
ln -s <MPI installation location>/bin/
mpirun.hp pbs_mpihp
```

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:

PBS_RSHCOMMAND=ssh

6.11.4 SGI MPI on the Altix Running ProPack 4 or 5

PBS supplies its own `mpiexec` on the Altix. This `mpiexec` uses the standard SGI `mpirun`. No unusual setup is required for either `mpiexec` or `mpirun,` however, there are prerequisites. See the following section. 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.

The PBS `mpiexec` is transparent to the user; MPI jobs submitted outside of PBS will run as they would normally. MPI jobs can be launched across multiple Altixes. PBS will manage, track, and cleanly terminate multi-host MPI jobs. PBS users can run MPI jobs within specific partitions.

If CSA has been configured and enabled, PBS will collect accounting information on all tasks launched by an MPI job. CSA information will be associated with the PBS job ID that invoked it, on each execution host. While each host involved in an MPI job will record CSA accounting information for the job if able to do so on the execution hosts, there is no tool to consolidate the accounting information from multiple hosts.

If the `PBS_MPI_DEBUG` environment variable's value has a nonzero length, PBS will write debugging information to standard output.

PBS uses the MPI-2 industry standard `mpiexec` interface to launch MPI jobs within PBS.

6.11.4.1 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 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.

6.11.4.2 Environment Variables

The PBS mpiexec script sets the PBS_CPUSET_DEDICATED environment variable to assert exclusive use of the resources in the assigned cpuset.

The PBS mpiexec checks the PBS_MPI_DEBUG environment variable. If this variable has a nonzero length, debugging information is written.

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.

The PBS mpiexec uses the value of PBS_O_PATH to search for the correct mpiexec if it was invoked by mistake.

6.11.5 SGI's MPI (MPT) Over InfiniBand

PBS jobs can run using SGI's MPI, called MPT, over InfiniBand. To use InfiniBand, set the MPI_USE_IB environment variable to 1.

6.11.6 The pbsrun_wrap Mechanism

PBS provides a mechanism for wrapping several versions/flavors of mpirun so that PBS can control jobs and perform accounting. PBS also provides a mechanism for unwrapping these versions of mpirun. The administrator wraps a version of mpirun using the pbsrun_wrap script, and unwraps it using the pbsrun_unwrap script. The pbsrun_wrap script is the installer script that wraps mpirun in a script called "pbsrun".

The `pbsrun_wrap` script instantiates the pbsrun script for each version of mpirun, renaming it to reflect the version/flavor of mpirun being wrapped. When executed inside a PBS job, the pbsrun script calls a version-specific initialization script which sets variables to control how the pbsrun script uses options passed to it. The pbsrun script uses `pbs_attach` to give MOM control of jobs.

The `pbsrun_wrap` command has a “-s” option. If `-s` is specified, then the "strict_pbs" options set in the various initialization scripts (e.g. pbsrun.ch_gm.init, etc...) will be set to 1 from the default 0. This means that the mpirun being wrapped by pbsrun will only get executed if inside a PBS environment. Otherwise, the user will get the error:

Not running under PBS
exiting since strict_pbs is enabled; execute only in PBS

The `pbsrun_wrap` command has this format:

pbsrun_wrap [-s] <path_to_actual_mpirun> pbsrun.<keyword>

If the mpirun wrapper script is run inside a PBS job, then it will translate 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).

If 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.

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.

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
```

The mpirun being wrapped must be installed and working on all the vnodes in the PBS cluster.

For all wrapped MPIs, the maximum number of ranks that can be launched in a job is the number of entries in the $PBS_NODEFILE.

6.11.6.1 The pbsrun Script

The pbsrun wrapper script is not meant to be executed directly but instead it is instantiated by `pbsrun_wrap`. It is copied to the target directory and renamed "pbsrun.<mpirun version/flavor>" where <mpirun version/flavor> is a string that identifies the mpirun version being wrapped (e.g. ch_gm).

The pbsrun script, if executed inside a PBS job, runs an initialization script, named $PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init, then parses mpirun-like arguments from the command line, sorting which options and option values to retain, to ignore, or to transform, before calling the actual mpirun script with a "pbs_attach" prefixed to the executable. The actual mpirun to call is found by tracing the link pointed to by $PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link.

6.11.6.2 The pbsrun Initialization Script

The initialization script, called $PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init, where <mpirun version/flavor> reflects the mpirun flavor/version being wrapped, can be modified by an administrator to customize against the local flavor/version of mpirun being wrapped. Inside this sourced init script, 8 variables are set:

```
options_to_retain="-optA -optB <val> -optC <val1> val2> ..."
options_to_ignore="-optD -optE <n> -optF <val1> val2> ..."
options_to_transform="-optG -optH <val> -optI <val1> val2> ..."
options_to_fail="-optY -optZ ..."
options_to_configfile="-optX <val> ..."
options_with_another_form="-optW <val> ..."
pbs_attach=pbs_attach
options_to_pbs_attach="-J $PBS_JOBID"
```

options_to_retain
: Space-separated list of options and values that pbsrun.<mpirun version/flavor> passes on to the actual mpirun call. options must begin with "-" or "--", and option arguments must be specified by some arbitrary name with left and right arrows, as in "<val1>".

options_to_ignore
: Space-separated list of options and values that pbsrun.<mpirun version/flavor> does not pass on to the actual mpirun call. Options must begin with "-" or "--", and option arguments must be specified by arbitrary names with left and right arrows, as in "<n>".

options_to_transform
: Space-separated list of options and values that pbsrun modifies before passing on to the actual mpirun call.

options_to_fail
: Space-separated list of options that will cause pbsrun to exit upon encountering a match.

options_to_configfile
: Single option and value that refers to the name of the "configfile" containing command line segments found in certain versions of mpirun.

options_with_another_form
: Space-separated list of options and values that can be found in options_to_retain, options_to_ignore, or options_to_transform, whose syntax has an alternate, unsupported form.

pbs_attach
: Path to pbs_attach, which is called before the <executable> argument of mpirun.

options_to_pbs_attach
: Special options to pass to the pbs_attach call. You may pass variable references (e.g. $PBS_JOBID) and they are substituted by pbsrun to actual values.

If pbsrun encounters any option not found in options_to_retain, options_to_ignore, and options_to_transform, then it is flagged as an error.

These functions are created inside the init script. These can be modified by the PBS administrator.

```
transform_action () {
     # passed actual values of
$options_to_transform
     args=$*
}

boot_action () {
     mpirun_location=$1
}

evaluate_options_action () {
     # passed actual values of transformed options
     args=$*
```

```
}

configfile_cmdline_action () {
     args=$*
}

end_action () {
     mpirun_location=$1
}
```

transform_action() The pbsrun.<mpirun version/flavor> wrapper script invokes the function transform_action() (called once on each matched item and value) with actual options and values received matching one of the "options_to_transform". The function returns a string to pass on to the actual mpirun call.

boot_action() Performs any initialization tasks needed before running the actual mpirun call. For instance, GM's MPD requires the MPD daemons to be user-started first. This function is called by the pbsrun.<mpirun version/flavor> script with the location of actual mpirun passed as the first argument. Also, the pbsrun.<mpirun version/flavor> checks for the exit value of this function to determine whether or not to progress to the next step.

evaluate_options_action()
Called with the actual options and values that resulted after consulting options_to_retain, options_to_ignore, options_to_transform, and executing transform_action(). This provides one more chance for the script writer to evaluate all the options and values in general, and make any necessary adjustments, before passing them on to the actual mpirun call. For instance, this function can specify what the default value is for a missing -np option.

configfile_cmdline_action()

Returns the actual options and values to be put in before the options_to_configfile parameter.

configfile_firstline_action()
: Returns the item that is put in the first line of the configuration file specified in the options_to_configfile parameter.

end_action()
: Called by pbsrun.<mpirun version/flavor> at the end of execution. It undoes any action done by transform_action(), like cleanup of temporary files. It is also called when pbsrun.<mpirun version/flavor> is prematurely killed. This function is called with the location of actual mpirun passed as first argument.

The actual mpirun program to call is the path pointed to by $PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link.

6.11.6.3 Modifying *.init Scripts

In order for administrators to modify *.init scripts without breaking package verification in RPM, master copies of the initialization scripts are named *.init.in. `pbsrun_wrap` instantiates the *.init.in files as *.init. For instance, `$PBS_EXEC/lib/MPI/pbsrun.mpich2.init.in` is the master copy, and `pbsrun_wrap` instantiates it as `$PBS_EXEC/lib/MPI/pbsrun.mpich2.init`. `pbsrun_unwrap` takes care of removing the *.init files.

6.11.6.4 Wrapping Multiple MPI's 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".

When you are done using them, unwrap them:

```
# pbsrun_unwrap pbsrun.mpich2_32
# pbsrun_unwrap pbsrun.mpich2_64
```

6.11.7 Wrapping MPICH-GM's mpirun.ch_gm with rsh/ssh

The PBS wrapper script to MPICH-GM's mpirun (mpirun.ch_gm) with rsh/ssh process startup method is named pbsrun.ch_gm. If executed inside a PBS job, this allows for PBS to track all MPICH-GM processes started by rsh/ssh so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard mpirun.ch_gm was used.

To wrap MPICH-GM's mpirun script:

```
pbsrun_wrap \
[MPICH-GM_BIN_PATH]/mpirun.ch_gm pbsrun.ch_gm
```

To unwrap MPICH-GM's mpirun script:

```
pbsrun_unwrap pbsrun.ch_gm
```

6.11.8 Wrapping MPICH-MX's mpirun.ch_gm with rsh/ssh

The PBS wrapper script to MPICH-MX's mpirun (mpirun.ch_gm) with rsh/ssh process startup method is named pbsrun.ch_mx. If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by rsh/ssh so that PBS can perform accounting and has complete job control. If executed outside of a PBS job, it behaves exactly as if standard mpirun.ch_mx was used.

To wrap MPICH-MX's mpirun script:

```
pbsrun_wrap \
[MPICH-MX_BIN_PATH]/mpirun.ch_mx
pbsrun.ch_mx
```

To unwrap MPICH-MX's mpirun script:

```
pbsrun_unwrap pbsrun.ch_mx
```

6.11.9 Wrapping MPICH-GM's mpirun.ch_gm with MPD

The PBS wrapper script to MPICH-GM's mpirun (mpirun.ch_gm) with MPD process startup method is called pbsrun.gm_mpd. If executed inside a PBS job, this allows for PBS to track all MPICH-GM processes started by the MPD daemons so that PBS can perform accounting have and complete job control. If executed outside of a PBS job, it behaves exactly as if standard mpirun.ch_gm with MPD was used.

To wrap MPICH-GM's mpirun script with MPD:

```
pbsrun_wrap \
MPICH-GM_BIN_PATH]/mpirun.mpd
pbsrun.gm_mpd
```

To unwrap MPICH-GM's mpirun script with MPD:

```
pbsrun_unwrap pbsrun.gm_mpd
```

6.11.10 MPICH-MX's mpirun.ch_mx with MPD

The PBS wrapper script to MPICH-MX's mpirun (mpirun.ch_mx) with MPD process startup method is called pbsrun.mx_mpd. If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by the MPD daemons so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard mpirun.ch_mx with MPD was used.

The script starts MPD daemons on each of the unique hosts listed in $PBS_NODEFILE, using either rsh or ssh method, based on value of environment variable RSHCOMMAND -- rsh is the default. The script also takes care of shutting down the MPD daemons at the end of a run.

To wrap MPICH-MX's mpirun script with MPD:

```
pbsrun_wrap \
[MPICH-MX_BIN_PATH]/mpirun.mpd
pbsrun.mx_mpd
```

To unwrap MPICH-MX's mpirun script with MPD:

```
pbsrun_unwrap pbsrun.mx_mpd
```

6.11.11 Wrapping MPICH2's mpirun

The PBS wrapper script to MPICH2's mpirun is called pbsrun.mpich2. If executed inside a PBS job, this allows for PBS to track all MPICH2 processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard MPICH2's mpirun was used.

The script takes care of ensuring that the MPD daemons on each of the host listed in the $PBS_NODEFILE are started. It also takes care of ensuring that the MPD daemons have been shut down at the end of MPI job execution.

To wrap MPICH2's mpirun script:

```
pbsrun_wrap [MPICH2_BIN_PATH]/mpirun \
pbsrun.mpich2
```

To unwrap MPICH2's mpirun script:

```
pbsrun_unwrap pbsrun.mpich2
```

In the case where MPICH2 uses mpirun.py, run pbsrun_wrap on mpirun.py itself.

6.11.12 Wrapping Intel MPI's mpirun

The PBS wrapper script to Intel MPI's mpirun is called pbsrun.intelmpi. If executed inside a PBS job, this allows for PBS to track all Intel MPI processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard Intel MPI's mpirun was used.

Intel MPI's mpirun itself takes care of starting/stopping the MPD daemons. pbsrun.intelmpi always passes the arguments -totalnum=<number of mpds to start> and -file=<mpd_hosts_file> to the actual mpirun, taking its input from unique entries in $PBS_NODEFILE.

To wrap Intel MPI's mpirun script:

```
pbsrun_wrap \
[INTEL_MPI_BIN_PATH]/mpirun pbsrun.intelmpi
```

To unwrap Intel MPI's mpirun script:

```
pbsrun_unwrap pbsrun.intelmpi
```

6.11.13 Wrapping MVAPICH1's mpirun

MVAPICH1 allows the use of InfiniBand. The PBS wrapper script to MVAPICH1's mpirun is called pbsrun.mvapich1. If executed inside a PBS job, this allows for PBS to track all MPI processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard MVAPICH1's mpirun was used.

If executed inside a PBS job script, all mpirun options given are passed on to the actual mpirun call with these exceptions:

-map <list>	The map option is ignored.
-exclude <list>	The exclude option is ignored.
-machinefile <file>	The machinefile option is ignored.

-np If not specified, the number of entries found in the $PBS_NODEFILE is used.

To wrap the MVAPICH1 mpirun:

```
pbsrun_wrap [MVAPICH1_BIN_PATH]/mpirun \
pbsrun.mvapich1
```

To unwrap MVAPICH1 mpirun:

```
pbsrun_unwrap pbsrun.mvapich1
```

6.11.14 Wrapping MVAPICH2's mpiexec

MVAPICH2 allows the use of InfiniBand. The PBS wrapper script to MVAPICH2's mpiexec is called pbsrun.mvapich2. If executed inside a PBS job, this allows for PBS to track all MPI processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard MVAPICH2's mpiexec had been used.

pbsrun.mvapich2 takes care of starting and stopping the MPD daemons if the user doesn't explicitly start and stop them.

If executed inside a PBS job script, all mpiexec options given are passed on to the actual mpiexec call with these exceptions:

-host <host> The host argument contents are ignored.

-machinefile <file> The file argument contents are ignored and replaced by the contents of the $PBS_NODEFILE.

To wrap the MVAPICH2 mpiexec:

```
pbsrun_wrap [MVAPICH2_BIN_PATH]/mpiexec \
pbsrun.mvapich2
```

To unwrap MVAPIC21 mpiexec:

```
pbsrun_unwrap pbsrun.mvapich2
```

6.11.15 Wrapping IBM's poe

MPI is supported under IBM's Parallel Operating Environment (POE) on AIX. Under AIX, the program `poe` is used to start user processes on remote machines. PBS will manage the IBM HPS in US (User Space) mode.

The PBS wrapper script to IBM's poe is called pbsrun.poe. If executed inside a PBS job, this allows for PBS to track all poe processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard IBM poe had been used.

If executed inside a PBS job script, all pbsrun.poe options given are passed on to standard poe with these exceptions:

-hostfile <file>	The file argument contents are ignored.
-procs <numranks>	If the -procs option or the MP_PROCS environment variable is not set by the user, a default of the number of entries in the file $PBS_NODEFILE is used.
-euilib {ip | us}	If the command line option -euilib is set, it will take precedence over the MP_EUILIB environment variable. If the -euilib option is set to us , user mode is set for the job. If the option is set to any other value, that value is passed to standard poe.
-msg_api	This option can only take the values "MPI" or "LAPI".

Environment Variables

MP_EUILIB	If the MP_EUILIB environment variable is set to us , user mode is set for the job. If the variable is set to any other value, that value is passed to standard poe.
MP_HOSTFILE	The MP_HOSTFILE environment variable is excised.

MP_PROCS — If the -procs option or the MP_PROCS environment variable is not set by the user, a default of the number of entries in the file $PBS_NODEFILE is used.

MP_MSG_API — This variable can only take the values "MPI" or "LAPI".

To wrap IBM poe:

```
pbsrun_wrap [POE_BIN_PATH]/poe pbsrun.poe
```

To unwrap IBM poe:

```
pbsrun_unwrap pbsrun.poe
```

You can use set the number of HPS US mode jobs MOM will accept:

Example: set node aix_15 to only accept one HPS US mode job at any one time:

```
# qmgr -c 'set node aix_15 \
resources_available.hps = 1'
```

Example: set node aix_75 to accept multiple HPS US mode jobs at any one time:

```
# qmgr -c 'set node aix_75 \
resources_available.hps = 99999'
```

You will need to set up a custom resource for the HPS so that hps is a static consumable host-level resource. See section 5.3.2 “Defining and Using a Custom Resource” on page 242. Users will need to request the “hps” resource in their select statements.

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 zero.

```
# qmgr -c 'set node not_ibm \
resources_available.hps = 0’
```

As an alternative, you can use "sharing=force_excl" to limit the number of HPS US mode jobs to 1, but it would be more restrictive. In this case, one and only one job could run on the HPS.

An example of the way to do this (in this case, changing the "sharing" attribute for a vnode named aix_15) uses the script “change_sharing”. See section 3.2.1 “Creation of Site-defined MOM Configuration Files” on page 109.

```
# cat change_sharing
$configversion 2
aix_15:  sharing = force_excl
# . /etc/pbs.conf
# $PBS_EXEC/sbin/pbs_mom -s insert \
   force_excl change_sharing
# pkill -HUP pbs_mom
```

6.12 SGI Job Container / Limits Support

PBS Professional supports the SGI Job Container/Limit feature. Each PBS job is placed in its own SGI Job container. Limits on the job are set as the `MIN(ULDB limit, PBS Resource_List limit)`. The ULDB domains are set in the following order:

PBS_{queue name}
PBS
batch

Limits are set for the following resources: `cput` and `vmem`. A job limit is *not* set for `mem` because the kernel does not factor in shared memory segments among `sproc`() processes, thus the system reported usage is too high.

For information on using Comprehensive System Accounting, see “Configuring MOM for Comprehensive System Accounting” on page 150.

6.13 Support for AIX

PBS Professional supports Large Page Mode on AIX. No additional steps are required from the PBS administrator. Certain applications (like many FEA Solvers) can benefit from using large page support. This allows programs to do considerably less page “thrashing”.

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.

6.14 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.

Each PBS user may submit several jobs at once. Each job may need to have data files staged to one or more execution hosts. 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.

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 `local_path@[remote_host:]remote_path`. The `local_path` is the path to the staging and execution directory. On stagein, `remote_path` is the path where the input files normally reside, and on stageout, `remote_path` is the path where output files will end up.

6.14.1 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 8: Effect of Job's sandbox Attribute on Location of Staging and Execution Directory

Job's sandbox attribute	Effect
not set	Job's staging and execution directory is the user's home directory
HOME	Job's staging and execution directory is the user's home directory
PRIVATE	Job's staging and execution directory is created under the directory specified in MOM $jobdir_root configuration option. If $jobdir_root is unset, it defaults to the user's home directory.

6.14.2 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_tm() 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.

Table 9: Options, Attributes, Environment Variables, etc., Affecting Staging

Option, Attribute, Environment Variable, etc.	Effect
MOM's `$jobdir_root` option	Directory under which PBS creates job-specific staging and execution directories. Defaults to user's home directory if unset. If `$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.
MOM's `$usecp` option	Tells MOM where to look for files in a shared file system; also tells MOM that she can use the local copy agent.
Job's `sandbox` attribute	Determines which directory PBS uses for the job's staging and execution. If value is `PRIVATE`, PBS uses a job-specific directory it creates under the location specified in the MOM `$jobdir_root` configuration option. If value is `HOME` or is unset, PBS uses the user's home directory for staging and execution. User-settable per-job via `qsub -W` or through a PBS directive. See the pbs_mom.8B man page.
Job's `stagein` attribute	Sets list of files or directories to be staged in. User-settable per job via `qsub -W`.
Job's `stageout` attribute	Sets list of files or directories to be staged out. User-settable per job via `qsub -W`.

Table 9: Options, Attributes, Environment Variables, etc., Affecting Staging

Option, Attribute, Environment Variable, etc.	Effect
Job's `jobdir` attribute	Set to pathname of staging and execution directory on primary execution host. Read-only; viewable via qstat -f.
Job's `Keep_Files` attribute	Determines whether output and/or error files remain on execution host. User-settable per job via `qsub -k` or through a PBS directive. If the `Keep_Files` attribute is set to `o` and/or `e` (output and/ or error files remain in the staging and execution directory) and the job's `sandbox` attribute is set to `PRIVATE`, standard out and/or error files are removed when the staging directory is removed at job end along with its contents.
Job's `PBS_JOBDIR` environment variable	Set to pathname of staging and execution directory on primary execution host. Added to environments of job script process, `pbs_tm` job tasks, prologue and epilogue, and MOM $action scripts.
Job's `TMPDIR` environment variable	Location of job-specific scratch directory.
`PBS_RCP` string in pbs.conf	Location of `rcp` command
`PBS_SCP` string in pbs.conf	Location of `scp` command; setting this parameter causes PBS to first try `scp` rather than `rcp` for file transport.
Server's `default_qsub_arguments` attribute	Can contain a default for job's `sandbox` (and other) attributes.

6.14.3 The Job's Lifecycle

6.14.3.1 Creation of TMPDIR

For each host allocated to the job, PBS creates a job-specific temporary scratch directory for this job. The location of TMPDIR is determined by MOM's `$tmpdir` configuration option. The TMPDIR environment variable is set to the pathname of the job-specific temporary scratch directory. 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.

6.14.3.2 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.

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 PBSEVENT_DEBUG3 level:
`"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 PBSEVENT_ERROR level:
`"unable to create the job directory <full path>"`.

When PBS creates a directory, the following message is logged at `PBSEVENT_JOB` level:
"created the job directory <full path>"

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`.

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.

6.14.3.3 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.

6.14.3.4 Staging Files Into Staging and Execution Directories

PBS evaluates `local_path` and `remote_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.

6.14.3.5 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` and `TMPDIR` set in its environment.

6.14.3.6 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.

6.14.3.7 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.

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.

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.

6.14.3.8 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` and `TMPDIR` set in its environment.

6.14.3.9 Staging Files Out and Removing Execution Directory

When PBS stages files out, it evaluates `local_path` and `remote_path` relative to `PBS_JOBDIR`. Files that cannot be staged out are saved in PBS_HOME/undelivered. See section 9.4.6 "Non Delivery of Output" on page 414.

When the job is done, PBS writes the final job accounting record and purges job information from the Server's database.

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.

User's Home Directory as Staging and Execution Directory

Files that are successfully staged out are deleted immediately, without regard to files that were not successfully staged out.

6.14.3.10 Removing TMPDIRs

PBS removes all TMPDIRs, along with their contents.

6.14.4 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.

6.14.5 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
```

6.15 Job Prologue / Epilogue Programs

PBS provides the ability for the Administrator to run a site-supplied script (or program) before (`prologue`) and/or after (`epilogue`) each job runs. This provides the capability to perform initialization or cleanup of resources, such as temporary directories or scratch files. The scripts may also be used to write "banners" on the job's output files. When multiple vnodes are allocated to a job, these scripts are run only by the "Mother Superior", the `pbs_mom` on the first vnode allocated. This is also where the job shell script is run. Note that both the `prologue` and `epilogue` are run under root (on UNIX) or an Admin-type account (on Windows), and neither is included in the job session, thus the `prologue` cannot be used to modify the job environment or change limits on the job. The prologue and epilogue run 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 primary purpose of the prologue is to provide a site with some means of performing addition checking prior to starting a job. The prologue can return values to indicate:

(0)	allow the job to continue to run
(1)	abort the job and discard it
(>1)	prevent the job from starting and requeue it

Note that the prologue does not have access to the $PBS_NODEFILE environment variable.

6.15.1 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

6.15.2 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 job's cpusets are destroyed

9 The epilogue is run

10 The obit is sent to the server

11 Any specified file staging out takes place, including stdout and stderr

12 Files staged in or out are removed

13 Any job-specific staging and execution directories are removed

14 Job files are deleted

15 FLEX licenses are returned to pool

If a `prologue` or `epilogue` script is not present, MOM continues in a normal manner. If present, the script is run with root/Administrator privilege. In order to be run, the script must adhere to the following rules:

- The script must be in the *PBS_HOME*/`mom_priv` directory with the exact name "`prologue`" (under UNIX) or "`prologue.bat`" (under Windows) for the script to be run before the job and the name "`epilogue`" (under UNIX) or "`epilogue.bat`" (under Windows) for the script to be run after the job.
- Under UNIX, 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.

The "script" may be either a shell script or an executable object file.

The `prologue` will be run prior to executing the job. When job execution completes for any reason (normal termination, job deleted while running, error exit, or even if pbs_mom detects an error and cannot completely start the job), the epilogue script will be run. If the job is deleted while it is queued, then neither the prologue nor the epilogue is run.

If a job is rerun or requeued as the result of being checkpointed, the exit status passed to the epilogue (and recorded in the accounting record) will have one of the following special values:

-11 - Job was rerun
-12 - Job was checkpointed and aborted

6.15.3 Prologue and Epilogue Arguments

When invoked, the prologue is called with the following arguments:

argv[1]	the job id.
argv[2]	the user name under which the job executes.
argv[3]	the group name under which the job executes.

The epilogue is called with the above, plus:

argv[4]	the job name.
argv[5]	the session id.
argv[6]	the requested resource limits (list).
argv[7]	the list of resources used
argv[8]	the name of the queue in which the job resides.
argv[9]	the account string, if one exists.
argv[10]	the exit status of the job.

For both the prologue and epilogue:

envp The environment passed to the script includes the contents of the pbs_environment file and PBS_JOBDIR.

cwd The current working directory is PBS_HOME/mom_priv (prologue) or the job's staging and execution directory (epilogue).

input When invoked, both scripts have standard input connected to a system dependent file. The default for this file is `/dev/null`.

output The standard output and standard error of the scripts are connected to the files which contain the standard output and error of the job. (Under UNIX, 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. Interactive jobs are only supported on UNIX.)

Important: Under Windows and with some UNIX shells, accessing **`arg[10]`** in the epilogue requires a shift in positional parameters. The script must call the arguments with indices 0 through 8, then perform a shift /8, then access the last argument using %9%. For example:

```
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
```

6.15.4 Prologue and Epilogue Time Out

When the scheduler runs a job it does not continue the cycle 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 sounds before the script has terminated, MOM will kill the script. The alarm value can be changed via $prologalarm MOM configuration parameter.

6.15.5 Prologue and Epilogue Error Processing

Normally, the prologue and epilogue programs should exit with a zero exit status. MOM will record in her log any case of a non-zero exit code. Exit status values and their impact on the job are:

Table 10:

Exit Code	Meaning	Prologue	Epilogue
-4	The script timed out (took too long).	The job will be requeued.	Ignored
-3	The wait(2) call waiting for the script to exit returned with an error.	The job will be requeued	Ignored
-2	The input file to be passed to the script could not be opened.	The job will be requeued.	Ignored
-1	The script has a permission error, is not owned by root, and/or is writable by others than root.	The job will be requeued.	Ignored
0	The script was successful.	The job will run.	Ignored
1	The script returned an exit value of 1.	The job will be aborted.	Ignored
>1	The script returned a value greater than one.	The job will be requeued.	Ignored

The above apply to normal batch jobs. Under UNIX, which supports interactive-batch jobs (qsub -I option), such jobs cannot be requeued on a non-zero status, and will therefore be aborted on any non-zero prologue exit.

Important: The Administrator must exercise great caution in setting up the `prologue` to prevent jobs from being flushed from the system.

`Epilogue` script exit values which are non-zero are logged, but have no impact on the state of the job. Neither prologue nor epilogue exit values are passed along as the job's exit value.

6.16 The Accounting Log

The PBS Server maintains an accounting log. 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
```

6.16.1 Accounting Log Format

The PBS accounting file is a text file with each entry terminated by a newline. The format of an entry is:

date time;record_type;id_string;message_text

The `date time` field is a date and time stamp in the format:

`mm/dd/yyyy hh:mm:ss`

The `id_string` is the job or reservation identifier. The `message_text` is ascii text. The content depends on the record type. The message text format is blank-separated keyword=value fields. The `record_type` is a single character indicating the type of record. The record types are:

`A` Job was aborted by the server.

`B` Beginning of reservation period. If the log entry is for a reservation, the `message_text` field contains information describing the specified reservation. Possible information includes:

Table 11: Reservation Information

Attribute	Explanation
owner=ownername	Name of party who submitted the reservation request.
name=reservation_name	If submitter supplied a name string for the reservation.
queue=queue_name	The name of the reservation queue.
ctime=creation_time	Time at which the reservation was created; seconds since the epoch.
start=period_start	Time at which the reservation period is to start, in seconds since the epoch.
end=period_end	Time at which the reservation period is to end, in seconds since the epoch.
duration= reservation_duration	The duration specified or computed for the reservation, in seconds.
exec_host=vnode_list	List of each vnode with vnode-level, consumable resources allocated from that vnode. exec_host=vnodeA/P*C [+vnodeB/P * C] where P is a unique index and C is the number of CPUs assigned to the reservation, 1 if omitted.
Authorized_Users= user_list	The list of users who are and are not authorized to submit jobs to the reservation.
Authorized_Groups= group_list	The list of groups who are and are not authorized to submit jobs to the reservation.
Authorized_Hosts= host_list	The list of hosts from which jobs may and may not be submitted to the reservation.

Table 11: Reservation Information

Attribute	Explanation
Resource_List= resources_list	List of resources requested by the reservation. Resources are listed individually as, for example: Resource_List.ncpus=16 Resource_List.mem=1048676.

`C` Job was checkpointed and held.

`D` Job was deleted by request. The `message_text` will contain `requester=user@host` to identify who deleted the job.

`E` Job ended (terminated execution). In this case, the `message_text` field contains information about 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 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. Possible information includes:

Table 12: PBS Job Information

Attribute	Explanation
`user=`username	The user name under which the job executed.
`group=`groupname	The group name under which the job executed.
`account=`account_string	If job has an “account name” string.
eligible_time	Amount of time job has waited while blocked on resources.
`jobname=`job_name	The name of the job.

Table 12: PBS Job Information

Attribute	Explanation
queue=queue_name	The name of the queue in which the job executed.
resvname=reservation_name	The name of the resource reservation, if applicable.
resvID=reservation_ID_string	The ID of the resource reservation, if applicable.
ctime=time	Time in seconds when job was created (first submitted).
qtime=time	Time in seconds when job was queued into current queue.
etime=time	Time in seconds when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues. Not affected by qaltering.
start=time	Time when job execution started.
exec_host=vnode_list	List of each vnode with vnode-level, consumable resources allocated from that vnode. exec_host=vnodeA/P*C [+vnodeB/P * C] where P is a unique index and C is the number of CPUs assigned to the job, 1 if omitted.
Resource_List.resource=amount	List of resources requested by the reservation. Resources are listed individually as, for example: Resource_List.ncpus=16 Resource_List.mem=1048676.
resources_used	Resources used by the job as reported by MOM. Typically includes ncpus, mem, vmem, cput, walltime, cpupercent.

Table 12: PBS Job Information

Attribute	Explanation
`session`=sessionID	Session number of job.
`alt_id`=id	Optional alternate job identifier. Included only for certain systems: On Altix machines with ProPack 4, the alternate id will show the path to the job's cpuset, starting with /PBSPro/.
`end`=time	Time in seconds since epoch when this accounting record was written.
`Exit_status`=value	The exit status of the job. See "Job Exit Codes" on page 410.
`resources_used.`*RES*=*value*	Provides the aggregate amount (*value*) of specified resource *RES* used during the duration of the job.
accounting_id=jidvalue	CSA JID, job container ID

Table 12: PBS Job Information

Attribute	Explanation
resource_assigned.RES= value	**Not** a job attribute; simply a label for reporting job resource assignment. The value of resources_assigned reported in the Accounting records is the actual amount assigned to the job by PBS. All allocated consumable resources will be included in the "resource_assigned" entries, one resource per entry. Consumable resources include ncpus, mem and vmem by default, and any custom resource defined with the -n or -f flags. A resource will not be listed if the job does not request it directly or inherit it by default from queue or server settings. For example, if a job requests one CPU on an Altix that has four 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.

F Resource reservation period finished.

K Scheduler or server requested removal of the reservation. The `message_text` field contains: `requester=Server@host` or `requester=Scheduler@host` to identify who deleted the resource reservation.

k Resource reservation terminated by ordinary client - e.g. an owner issuing a `pbs_rdel` command. The `message_text` field contains: `requester=user@host` to identify who deleted the resource reservation.

`L` License information. This line in the log will contain the following fields:
Log date; record type; keyword; specification for floating license; hour; day; month; max
The following table explains each field:

Table 13: Licensing Info in Accounting Log

Field	Explanation
Log date	Date of event
record type	Indicates license info
keyword	license
specification for floating license	Indicates that this is floating license info
hour	Number of licenses used in the last hour
day	Number of licenses used in the last day
month	Number of licenses used in the last month
max	Maximum number of licenses ever used. Not dependent on server restarts.

`Q` Job entered a queue. For this kind of record type, the message_text contains `queue=`name identifying the queue into which the job was placed. There will be a new Q record each time the job is routed or moved to a new (or the same) queue.

`R` Job was rerun.

S Job execution started. The message_text field contains:

Table 14:

Attribute	Explanation
user=username	The user name under which the job will execute.
group=groupname	The group name under which the job will execute.
jobname=job_name	The name of the job.
queue=queue_name	The name of the queue in which the job resides.
ctime=time	Time in seconds when job was created (first submitted).
qtime=time	Time in seconds when job was queued into current queue.
etime=time	Time in seconds when job became eligible to run; no holds, etc.
start=time	Time in seconds when job execution started.
exec_host=vnode_list	List of each vnode with vnode-level, consumable resources allocated from that vnode. exec_host=vnodeA/P*C [+vnodeB/P * C] where P is the job number and C is the number of CPUs assigned to the job, 1 if omitted.
resource_assigned	**Not** a job attribute; instead simply a label for reporting resources assigned to a job. Consumable resources that were allocated to that job.
Resource_List.resource= amount	List of resources requested by the reservation. Resources are listed individually as, for example: Resource_List.ncpus=16 Resource_List.mem=1048676.

Table 14:

Attribute	Explanation
session=sessionID	Session number of job.
accounting_id= identifier_string	An identifier that is associated with system-generated accounting data. In the case where accounting is CSA on Altix, identifier_string is a job container identifier or JID created for the PBS job.

T Job was restarted from a checkpoint file.

U Created unconfirmed reservation on Server. The message_text field contains requester=user@host to identify who requested the reservation.

Y Reservation confirmed by the Scheduler. The message_text field contains requester=user@host to identify who requested the reservation.

For Resource_List and resources_used, there is one entry per resource, corresponding to the resources requested and used, respectively.

Important: If a job ends between MOM poll cycles, resources_used.*RES* numbers will be slightly lower than they are in reality. For long-running jobs, the error percentage will be minor.

6.16.2 PBS Accounting and Windows

PBS will save information such as user name, 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:

1. Bring up the `Start Menu->Control Panel ->Performance and Maintenance->Administrative Tools->Services` dialog box (Windows XP).
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

Important: 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”

6.17 Use and Maintenance of Logfiles

The PBS system tends to produce a large number of logfile entries. There are two types of logfiles: the event logs which record events from each PBS component (`pbs_server`, `pbs_mom`, and `pbs_sched`) and the PBS accounting log.

6.17.1 PBS Events

The amount 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. The specified events are logically “or-ed” to produce a mask representing the events the local site wishes to have logged. (Note that this is opposite to the scheduler’s log filters, which specify what to leave out.) The available events, and corresponding decimal and hexadecimal values are shown below. When these appear in the log file, they are tagged with the hexadecimal shown, without a preceding “0x”.

Table 15: PBS Events

Value	Hex	Event Description
1	0001	Internal PBS errors.
2	0002	System (OS) errors, such as malloc failure.
4	0004	Administrator-controlled events, such as changing queue attributes.
8	0008	Job related events: submitted, ran, deleted, ...
16	0010	Job resource usage.
32	0020	Security related, e.g. attempts to connect from an unknown host.
64	0040	When the Scheduler was called and why.
128	0080	Debug messages. Common messages..
256	0100	Debug level 2.
512	0200	Reservation-related messages
1024	0400	Debug level 3. Most prolific debug messages

For example, if you want to log all events except those at levels 512 and 1024 (hex 0x200 and 0x400), you would use a log level of 511. This is 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1. If you want to log events at levels 1, 2, and 16, you would set the log level to 19.

The event logging mask is controlled differently for the different components. The following table shows the log event parameter for each, and page reference for details.

Table 16:

PBS	Attribute and Reference	Notes
Server	See “log_events” on page 24.	Takes effect immediately with `qmgr`
MOM	See “$logevent <mask>” on page 117.	Requires `SIGHUP` to MOM
Scheduler	See “log_filter” on page 167.	Requires `SIGHUP` to Scheduler

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. Appendix B contains a listing of all “types” and each corresponding batch request.

6.17.1.1 Scheduler Commands

These commands provide the scheduler a hint as to why a scheduling cycle is being started. The following table shows commands from the server to the scheduler.

Table 17: Commands from Scheduler to Server

Value	Event Description
1	New job enqueued
2	Job terminated
3	Scheduler time interval reached
4	Cycle again after scheduling one job
5	Scheduling command from operator or manager
7	Configure

Table 17: Commands from Scheduler to Server

Value	Event Description
8	Quit (qterm -s)
9	Ruleset changed
10	Schedule first
11	A reservation's start time has been reached
12	Schedule a job (qrun command has been given)

6.17.2 Event Logfiles

Each PBS component maintains separate event logfiles. The logfiles default to a file with the current date as the name in the `PBS_HOME/(component)_logs` directory. This location can be overridden with the "`-L pathname`" option where pathname must be an absolute path.

The log filters work differently: the server and MOM log filters specify what to put in the log file, and the scheduler's log filter specifies what to keep out of its log files.

If the default logfile name is used (no `-L` option), the log will be closed and reopened with the current date daily. This happens on the first message after midnight. If a path is given with the `-L` option, the automatic close/ reopen does not take place.

On UNIX, all components will close and reopen the same named log file on receipt of **SIGHUP**. The process identifier (PID) of the component is available in its lock file in its home directory. Thus it is possible to move the current log file to a new name and send **SIGHUP** to restart the file thusly:

```
cd $PBS_HOME/component_logs
mv current archive
kill -HUP \
   'cat ../component_priv/component.lock'
```

On Windows, manual rotation of the event log files can be accomplished by stopping the particular PBS service component for which you want to rotate the logfile, moving the file, and then restarting that component. For example:

```
cd "%PBS_HOME%\component_logs"
net stop pbs_component
move current archive
net start pbs_component
```

Each daemon will write 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 pbs_version information and build information will appear in individual records. These records will contain the 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=KCRYPT ...

6.17.3 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:

date-time;event_code;server_name;object_type;object_name;message

The `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 6.17.1 "PBS Events" on page 354 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.

All messages are associated with an `object_type`, where the `object_type` is the type of object which the message is about. The following lists each possible `object_type`:

Svr	for server
Que	for queue
Job	for job
Req	for request
Fil	for file
Act	for accounting string
Node	for vnode or host
Resv	for reservation
Sched	for scheduler

The `object_name` is the name of the specific object. `message_text` field is the text of the log message.

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 logging level of these messages is PBSEVENT_DEBUG2.

To append job usage to standard output for an interactive job, use a shell script for the epilogue which contains the following:

```
#!/bin/sh
tracejob -sl $1 | grep 'cput'
```

6.18 Using the UNIX syslog Facility

Each PBS component logs various levels 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 18:

PBS_LOCALLOG=x	Enables logging to local PBS log files. Only possible when logging via syslog feature is enabled. 0 = no local logging 1 = local logging enabled
PBS_SYSLOG=x	Controls the use of syslog and syslog "facility" under which the entries are logged. If x is: 0 - no syslogging 1 - logged via LOG_DAEMON facility 2 - logged via LOG_LOCAL0 facility 3 - logged via LOG_LOCAL1 facility ... 9 - logged via LOG_LOCAL7 facility
PBS_SYSLOGSEVR=y	Controls the severity level of messages that are logged; see /usr/include/sys/syslog.h. If y is: 0 - only LOG_EMERG messages are logged 1 - messages up to LOG_ALERT are logged ... 7 - messages up to LOG_DEBUG are logged

Important: PBS_SYSLOGSEVR is used in addition to PBS's log_events mask which controls the class of events (job, vnode, ...) that are logged.

6.19 Managing Jobs

6.19.1 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:

+ 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.

- 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.

6.19.2 Managing Jobs on Machines with cpusets

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.

6.19.3 Job IDs

The largest possible job ID is the 7-digit number 9999999. After this has been reached, job IDs start again at zero.

6.19.4 Job States

Job states are abbreviated to one character.

Table 19: Job States

State	Description
B	Job arrays only: job array has started
E	Job is exiting after having run
H	Job is held. A job is put into a held state by the server or by a user or administrator. A job stays in a held state until it is released by a user or administrator.
Q	Job is queued, eligible to run or be routed
R	Job is running
S	Job is suspended by server. A job is put into the suspended state when a higher priority job needs the resources.
T	Job is in transition (being moved to a new location)
U	Job is suspended due to workstation becoming busy
W	Job is waiting for its requested execution time to be reached or job specified a stagein request which failed for some reason.
X	Subjobs only; subjob is finished (expired.)

6.19.4.1 Job Substates

Table 20: Job Substates

Substate Number	Substate Description
00	Transit in, prior to waiting for commit
01	Transit in, waiting for commit
02	transiting job outbound, not ready to commit

Table 20: Job Substates

Substate Number	Substate Description
03	transiting outbound, ready to commit
10	job queued and ready for selection
11	job queued, has files to stage in
14	job staging in files before waiting
15	job staging in files before running
16	job stage in complete
20	job held - user or operator
22	job held - waiting on dependency
30	job waiting until user-specified execution time
37	job held - file stage in failed
41	job sent to MOM to run
42	Running
43	Suspended by Operator or Manager
44	job sent to run under Globus
45	Suspended by Scheduler
50	Server received job obit
51	Staging out stdout/err and other files
52	Deleting stdout/err files and staged-in files
53	Mom releasing resources
54	job is being aborted by server
56	(Set by MOM) Mother Superior telling sisters to kill everything
57	(Set by MOM) job epilogue running

Table 20: Job Substates

Substate Number	Substate Description
58	(Set by MOM) job obit notice sent
59	Waiting for site "job termination" action script
60	Job to be rerun, MOM sending stdout/stderr back to Server
61	Job to be rerun, staging out files
62	Job to be rerun, deleting files
63	Job to be rerun, freeing resources
70	Array job has begun
153	(Set by MOM) Mother Superior waiting for delete ACK from sisters

6.19.5 Where to Find Job Information

Information about jobs is found in PBS_HOME/server_priv/jobs and PBS_HOME/mom_priv/jobs.

If PBS tries to requeue a job and cannot, for example when the queue doesn't exist, the job is deleted.

Chapter 7

Administrator Commands

There are two types of commands in PBS: those that users use to manipulate their own jobs, and those that the PBS Administrator uses to manage the PBS system. This chapter covers the various PBS administrator commands.

The table below lists all the PBS commands; the left column identifies all the user commands, and the right column identifies all the administrator commands. (The user commands are described in detail in the **PBS Professional User's Guide**.)

Individuals with PBS Operator or Manager privilege can use the user commands to act on any user job. For example, a PBS Operator can delete or move any user job. (Detailed discussion of privilege within PBS is discussed under the heading of section 6.8.7 "External Security" on page

302.)

Some of the PBS commands are intended to be used only by the PBS Operator or Manager. These are the administrator commands, which are described in detail in this chapter. Some administrator commands can be executed by normal users but with limited results. The `qmgr` command can be run by a normal user, who can view but cannot alter any Server configuration information. If you want normal users to be able to run the `pbs-report` command, you can add read access to the `server_priv/accounting` directory, enabling the command to report job-specific information. Be cautioned that all job information will then be available to all users. Likewise, opening access to the accounting records will permit additional information to be printed by the `tracejob` command, which normal users would not have permissions to view. In either case, an administrator-type user (or UNIX root) always has read access to these data.

Most commands, when given the sole option "--version", will output the version of PBS for that command and exit. For example,

`qmgr --version`

will cause the **`qmgr`** command to output version information and exit. See each command's manual page.

Under Windows, use double quotes when specifying arguments to PBS commands.

Table 1: PBS Professional User and Manager Commands

User Commands		Administrator Commands	
Command	Purpose	Command	Purpose
nqs2pbs	Convert from NQS	pbs-report	Report job statistics
pbs_rdel	Delete Reservation		
pbs_rstat	Status Reservation	pbs_hostn	Report host name(s)

Table 1: PBS Professional User and Manager Commands

User Commands		Administrator Commands	
pbs_ password	Update per user / per server password[1]	pbs_migrate _users	Migrate per user / per server pass- words [1]
pbs_rsub	Submit Reservation	pbs_probe	PBS diagnostic tool
pbsdsh	PBS distributed shell	pbs_rcp	File transfer tool
qalter	Alter job	pbs_tclsh	TCL with PBS API
qdel	Delete job	pbsfs	Show fairshare usage
qhold	Hold a job	pbsnodes	Node manipula- tion
qmove	Move job	printjob	Report job details
qmsg	Send message to job	qdisable	Disable a queue
qorder	Reorder jobs	qenable	Enable a queue
qrls	Release hold on job	qmgr	Manager inter- face
qselect	Select jobs by crite- ria	qrerun	Requeue running job
qsig	Send signal to job	qrun	Manually start a job
qstat	Status job, queue, Server	qstart	Start a queue
qsub	Submit a job	qstop	Stop a queue
tracejob	Report job history	qterm	Shut down PBS
xpbs	Graphical User Inter- face	xpbsmon	GUI monitoring tool

Notes: 1 Available on Windows only.

7.1 The pbs_hostn Command

The **`pbs_hostn`** command takes a hostname, and reports the results of both `gethostbyname(3)` and `gethostbyaddr(3)` system calls. Both forward and reverse lookup of hostname and network addresses need to succeed in order for PBS to authenticate a host and function properly. Running this command can assist in troubleshooting problems related to incorrect or non-standard network configuration, especially within clusters. The command usage is:

pbs_hostn [-v] hostname

The available options, and description of each, follows.

Table 2:

Option	Description
-v	Turns on verbose mode

7.2 The pbs_migrate_users Command

During a migration upgrade in Windows environments, if the Server attribute `single_signon_password_enable` is set to "true" in both the old Server and the new Server, the per-user/per-server passwords are not automatically transferred from an old Server to the new Server. The `pbs_migrate_users` command is provided for migrating the passwords. (Note that users' passwords on the old Server are not deleted.) The command usage is:

pbs_migrate_users *old_server*[:port] *new_server*[:port]

The exit values and their meanings are:

0 success

-1 writing of passwords to files failed.
-2 communication failures between old Server and new Server
-3 `single_signon_password_enable` not set in either old Server or new Server.
-4 the current user is not authorized to migrate users

7.3 The pbs_rcp vs. scp Command

The **pbs_rcp** command is used internally by PBS as the default file delivery mechanism. PBS can be directed to use Secure Copy (**scp**) by so specifying in the PBS global configuration file. Specifically, to enable `scp`, set the `PBS_SCP` parameter to the full path of the local `scp` command, as described in the discussion of "pbs.conf" on page 274.) This should be set on all vnodes where there is or will be a PBS MOM running. MOMs already running will need to be stopped and restarted.

7.4 The pbs_probe Command

The **pbs_probe** command reports post-installation information that is useful for PBS diagnostics. Aside from the direct information that is supplied on the command line, `pbs_probe` reads basic information from the `pbs.conf` file, and the values of any of the following environment variables that may be set in the environment in which `pbs_probe` is run: `PBS_CONF`, `PBS_HOME`, `PBS_EXEC`, `PBS_START_SERVER`, `PBS_START_MOM`, and `PBS_START_SCHED`.

Important: The `pbs_probe` command is currently only available on UNIX; in Windows environments, use the `pbs_mkdirs` command instead.

The `pbs_probe` command usage is:

pbs_probe [-f | -v]

If no options are specified, `pbs_probe` runs in "report" mode, in which it will report on any errors in the PBS infrastructure files that it detects. The problems are categorized, and a list of the problem messages in each category are output. Those categories which are empty do not show in the output.

The available options, and description of each, follows.

Table 3:

Option	Description
-f	Run in “fix” mode. In this mode `pbs_probe` will examine each of the relevant infrastructure files and, where possible, fix any errors that it detects, and print a message of what got changed. If it is unable to fix a problem, it will simply print a message regarding what was detected.
-v	Run in “verbose” mode. If the verbose option is turned on, `pbs_probe` will also output a complete list of the infrastructure files that it checked.

7.5 The pbsfs (PBS Fairshare) Command

The **`pbsfs`** command allows the Administrator to display or manipulate PBS fairshare usage data. The `pbsfs` command can only be run as root (UNIX) or a user with Administrator privilege (Windows). If the command is to be run with options to alter/update the fairshare data, the Scheduler must not be running. If you terminate the Scheduler, be sure to restart it after using the `pbsfs` command.

For printing, the scheduler can be running, but the data may be stale. To make sure the data isn't stale when being printed, sending a kill -HUP to the scheduler will force the scheduler to write out its internal cache.

Important: If the Scheduler is killed, it will lose any new fairshare data since the last synchronization. For suggestions on minimizing or eliminating possible data loss, see section 4.16.11 “Viewing and Managing Fairshare Data” on page 225.

The command usage is:

```
pbsfs [ -d | -e | -p | -t ]
pbsfs [ -c entity1 entity2 ] [ -g entity ]
    [ -s entity usage_value ]
```

The available options, and description of each, follows.

Table 4:

Option	Description	Scheduler: Up/Down
-c *entity1 entity2*	Compare two *entities* and print the most deserving entity.	Up
-d	Decay the fairshare tree (divide all values in half)	Down
-e	Trim fairshare tree to include only entries in `resource_group` file	Down
-g *entity*	Print all data for *entity* and path from the root of tree to node.	Up
-p	Print the fairshare tree in a flat format (default format).	Up
-s *entity usage_value*	Set *entity's* usage value to *usage_value*. Note that editing a non-leaf node is ignored. All non-leaf usage values are calculated each time the Scheduler is run or HUPed.	Down
-t	Print the fairshare tree in a hierarchical format.	Up

There are multiple parts to a fairshare node and you can print these data in different formats.The data displayed is:

Table 5:

Data	Description
entity	the name of the entity to use in the fairshare tree
group	the group ID the entity is in (i.e. the entity's parent)
cgroup	the group ID of this entity
shares	the number of shares the entity has

Table 5:

usage	the amount of usage
percentage	the percentage the entity has of the tree. Note that only the leaves sum to 100%. If all of the nodes are summed, the result will be greater then 100%. Only the leaves of the tree are fairshare entities.
usage / perc	The value the Scheduler will use to pick which entity has priority over another. The smaller the number the higher the priority.
path from root	The path from the root of the tree to the leaf node. This is useful because the Scheduler will compare two entities by starting at the root, and working toward the leaves, to determine which has the higher priority.
resource	Resource for which usage is accumulated for the fairshare calculations. Default is `cput` (CPU seconds) but can be changed in `sched_config`.

Whenever the fairshare usage database is changed, the original database is saved with the name "`usage.bak`". Only one backup will be made.

Subjobs are treated as regular jobs in the case of fairshare. Fairshare data may not be accurate for job arrays, because subjobs are typically shorter than the scheduler cycle, and data for them can be lost.

7.5.1 Trimming the Fairshare Data

Fairshare usage data may need to be trimmed because of the way the scheduler deals with unknown entities which have usage data. If the scheduler finds an entity which has usage data, but is not in the resource_group file, it will add it to the "unknown" group. This is sometimes the result of a typo. It will also be be what happens to accounts that are no longer in a group. Trimming the fairshare tree is a good way to get rid of these.

The recommended set of steps to use `pbsfs` to trim fairshare data are as follows:

- First send a HUP signal to the Scheduler to force current fairshare usage data to be written, then terminate the Scheduler:

  UNIX:

  kill -HUP *pbs_sched_PID*

  kill *pbs_sched_PID*

  Windows:

  net stop pbs_sched

- Now you can modify the `$`*`PBS_HOME`*`/sched_priv/resource_group` file if needed. When satisfied with it, run the `pbsfs` command to trim the fairshare tree:

  pbsfs -e

- Lastly, restart the Scheduler:

  UNIX:

  $*PBS_EXEC*/sbin/pbs_sched

  Windows:

  net start pbs_sched

7.6 The pbs_tclsh Command

The **pbs_tclsh** command is a version of the TCL shell (tclsh) linked with special TCL-wrapped versions of the PBS Professional external API library calls. This enables the user to write TCL scripts which utilize the PBS Professional API to query information. For usage see the `pbs_tclapi(3B)` manual page, and the **PBS Professional External Reference Specification.**

The `pbs_tclsh` command is supplied with the standard PBS binary. Users can make queries of MOM using this utility, for example:

% **pbs_tclsh**
tclsh> **openrm <hostname>**
<fd>
tclsh> **addreq <fd> "loadave"**
tclsh> **getreq <fd>**
5.0
tclsh> **closereq <fd>**

7.7 The pbsnodes Command

The **`pbsnodes`** command is used to query the status of hosts, or to mark hosts OFFLINE or FREE. The `pbsnodes` command obtains host information by sending a request to the PBS server.

To print the status of the specified host(s), run `pbsnodes` without options and with a list of hosts (and optionally the `-s` option.)

To print the command usage, run `pbsnodes` with no options and no arguments.

When the pbsnodes command is run with Manager or Operator privilege, it will output version information for each node specified by the command.

PBS Manager or Operator privilege is required to execute `pbsnodes` with the `-c` , `-o` , or `-r` options.

To remove a host from the scheduling pool, mark it OFFLINE. If a node has been marked DOWN, the server will mark it FREE the next time it can contact the MOM.

For hosts with multiple vnodes, `pbsnodes` operates on a host and all of its vnodes, where the hostname is resources_available.host. See the `-v` option.

Custom resources can be made invisible to users or unalterable by users via resource permission flags. See section 2.10.9 “Resource Flags for Resource Permissions” on page 71. A user will not be able to view a host-level custom resource which has been made either invisible or unalterable.

To act on vnodes, use the `qmgr` command.

Syntax:

pbsnodes [-c | -o | -r] [-s server]
hostname [hostname ...]

pbsnodes [-l] [-s server]

pbsnodes -a [-v] [-s server]

Options::

Table 6:

Option	Description
(no options)	If neither options nor a host list is given, the `pbsnodes` command prints usage syntax.
-a	Lists all hosts and all their attributes (available and used.) When listing a host with multiple vnodes: 1. The output for the jobs attribute lists all the jobs on all the vnodes on that host. Jobs that run on more than one vnode will appear once for each vnode they run on. 2. For consumable resources, the output for each resource is the sum of that resource across all vnodes on that host. 3. For all other resources, e.g. string and boolean, if the value of that resource is the same on all vnodes on that host, the value is returned. Otherwise the output is the literal string `"<various>"`.
-c host list	Clears OFFLINE and DOWN from listed hosts. The listed hosts will become FREE if they are online, or remain DOWN if they are not (for example, powered down.) Requires PBS Manager or Operator privilege.
host list	Prints information for the specified host(s).

Table 6:

Option	Description
-l	Lists all hosts marked as DOWN or OFFLINE. Each such host's state and comment attribute (if set) is listed. If a host also has state STATE-UNKNOWN, that will be listed. For hosts with multiple vnodes, only hosts where all vnodes are marked as DOWN or OFFLINE are listed.
-o host list	Marks listed hosts as OFFLINE even if currently in use. This is different from being marked DOWN. A host that is marked OFFLINE will continue to execute the jobs already on it, but will be removed from the scheduling pool (no more jobs will be scheduled on it.) Requires PBS Manager or Operator privilege.
-r host list	Clears OFFLINE from listed hosts.
-s server	Specifies the PBS *`server`* to which to connect.
-v	Can only be used with the **`-a`** option. Prints one entry for each vnode in the PBS complex. (Information for all hosts is displayed.) The output for the jobs attribute for each vnode lists the jobs executing on that vnode. The output for resources and attributes lists that for each vnode.

7.8 The printjob Command

The **`printjob`** command is used to print the contents of the binary file representing a PBS batch job saved within the PBS system. By default all the job data including job attributes are printed. This command is useful for troubleshooting, as during normal operation, the `qstat` command is the preferred method for displaying job-specific data and attributes. The command usage is:

printjob [-a] file [file...]

The available options, and description of each, follows.

Table 7:

Option	Description
-a	Suppresses the printing of job attributes.

7.9 The tracejob Command

PBS includes the **`tracejob`** utility to extract daemon/service logfile messages for a particular job (from all log files available on the local host) and print them sorted into chronological order.

Important: By default a normal user does not have access to the accounting records, and so information contained therein will not be displayed. However, if an administrator or UNIX root runs the tracejob command, this data will be included.

Usage for the `tracejob` command is:

tracejob [-a|s|l|m|v][-w cols][-p path][-n days][-f filter]
[-c count] jobid

Note: for an array job, the job ID must be enclosed in double quotes.

The available options, and description of each, follows.

Table 8: Tracejob Options

Option	Description
-a	Do not report accounting information.
-c <count>	Set excessive message limit to <u>count</u>. If a message is logged at least <u>count</u> times, only the most recent message is printed. Default for <u>count</u> is 15.

Table 8: Tracejob Options

Option	Description
-f <filter>	Do not include logs of type filter. The -f option can be used more than once on the command line. filter: error, system, admin, job, job_usage, security, sched, debug, debug2
-l	Do not report scheduler information.
-m	Do not report MOM information.
-n <days>	Report information from up to days days in the past. Default is 1 = today.
-p <path>	Use path as path to PBS_HOME on machine being queried.
-s	Do not report server information.
-w <cols>	Width of current terminal. If not specified by the user, tracejob queries OS to get terminal width. If OS doesn't return anything, default is 80.
-v	Verbose. Report more of tracejob’s errors than default.
-z	Disable excessive message limit. Excessive message limit is enabled by default.

For more information, see man(8) tracejob.

The following example requests all log messages for a particular job from today’s (the default date) log file. Note that the third column of the display contains a single letter (S, M, A, or L) indicating the source of the log message (**S**erver, **M**OM, **A**ccounting, or schedu**L**er log files).

7.9.1 Example Tracejob Output

`tracejob 420`

```
Job: 420.myhost

04/24/2008 11:35:41  L   Considering job to run
04/24/2008 11:35:41  S   Job Modified at request of
                                   Scheduler@myhost.mydomain.com
04/24/2008 11:35:41  S   Job Queued at request of
                                   user1@myhost.mydomain.com,
owner =
                                   user1@myhost.mydomain.com, job
                                   name = ExampleJob, queue = workq
04/24/2008 11:35:41  S   Job Run at request of
                                   Scheduler@myhost.mydomain.com
on
                                   hosts (myhost:ncpus=1)
04/24/2008 11:35:41  M    Started, pid = 6802
04/24/2008 11:35:41  S   enqueuing into workq, state 1 hop 1
04/24/2008 11:35:41  L   Job run
04/24/2008 11:36:41  M    Obit sent
04/24/2008 11:36:41  M    task 00000001 cput= 0:00:00
04/24/2008 11:36:41  M    Terminated
04/24/2008 11:36:41  M    myhost cput= 0:00:00 mem=1640kb
04/24/2008 11:36:41  S   Obit received
04/24/2008 11:36:41  M    task 00000001 terminated
04/24/2008 11:36:41  M    kill_job
04/24/2008 11:36:41  S   Exit_status=0
                                   resources_used.cpupercent=0
                                   resources_used.cput=00:00:00
                                   resources_used.mem=1640kb
                                   resources_used.ncpus=1
                                   resources_used.vmem=1831788kb
                                   resources_used.walltime=00:01:00
04/24/2008 11:36:42  S   dequeuing from workq, state 5
```

7.10 The qdisable Command

The `qdisable` command directs that the designated queue should no longer accept batch jobs. If the command is successful, the queue will no longer accept Queue Job requests which specified the now-disabled queue. Jobs which already reside in the queue will continue to be processed. This allows a queue to be “drained.” The command usage is:

qdisable destination ...

7.11 The qenable Command

The `qenable` command directs that the designated queue should accept batch jobs. This command sends a Manage request to the batch Server specified on the command line. If the command is accepted, the now-enabled queue will accept Queue Job requests which specify the queue. The command usage is:

qenable destination ...

7.12 The qstart Command

The `qstart` command directs that the designated queue should process batch jobs. If the queue is an execution queue, the Server will begin to schedule jobs that reside in the queue for execution. If the designated queue is a routing queue, the Server will begin to route jobs from that queue. The command usage is:

qstart destination ...

7.13 The qstop Command

The `qstop` command directs that the designated queue should stop processing batch jobs. If the designated queue is an execution queue, the Server will cease scheduling jobs that reside in the queue for execution. If the queue is a routing queue, the Server will cease routing jobs from that queue. The command usage is:

qstop destination ...

7.14 The qrerun Command

The **`qrerun`** command directs that the specified jobs are to be rerun if possible. To rerun a job is to terminate the session leader of the job and return the job to the queued state in the execution queue in which the job currently resides. If a job is marked as not rerunnable then the rerun request will fail for that job. (See also the discussion of the `-r` option to `qsub` in the **PBS Professional User's Guide**.) The command usage is:

qrerun [-W force] jobID [jobID ...]

Note: for array jobs, the job IDs must be enclosed in double quotes.

The available options, and description of each, follows.

Table 9:

Option	Description
-W force	This option, where `force` is the literal character string "force", directs that the job is to be requeued even if the vnode on which the job is executing is unreachable.

The qrerun command can be used on a job array, a subjob, or a range of subjobs. If the qrerun command is used on a job array, all of that array's currently running subjobs and all of its completed and deleted subjobs are requeued.

7.15 The qrun Command

The **`qrun`** command is used to force a Server to initiate the execution of a batch job. The job can be run regardless of scheduling position, resource requirements and availability, or state; see the -H option. You can overload CPUs using this command. The command usage is:

qrun [-a] [-H host-spec] jobID [jobID ...]

Note: for array jobs, some shells require that job IDs be enclosed in double quotes.

The available options, and description of each, follows.

Table 10:

Option	Description
-a	Specifies that the `qrun` command will exit before the job actually starts execution.
-H host-spec	Specifies the vnode(s) within the complex on which the job(s) are to be run. The *host-spec* argument is a plus-separated list of vnode names, e.g. VnodeA+VnodeB+VnodeC. Resources can be specified in this fashion: VnodeA:mem=100kb:ncpus=1+VnodeB:mem=100kb:ncpus=2

See section 4.3.1 “Rules for Submitting Jobs” on page 42 of the **PBS Professional User’s Guide** for detailed information on requesting resources and placing jobs on vnodes.

No -H hosts option

If the operator issues a qrun request of a job without -H hosts, the server will make a request of the scheduler to run the job immediately. The scheduler will run the job if the job is otherwise runnable by the scheduler:

The queue in which the job resides is an execution queue
and is started.

The job is in the queued state.

Either the resources required by the job are available, or preemption is enabled and the required resources can be made available by preempting jobs that are running.

-H hosts option	If the -H hosts option is used, the Server will immediately run the job on the named hosts, regardless of current usage on those vnodes.
-H hosts option with list of vnodes	If a “+” separated list of hosts is specified in the Run Job request, e.g. VnodeA+VnodeB+... the Scheduler will apply one requested chunk from the select directive in round-robin fashion to each vnode in the list.
-H hosts option with list of vnodes and resource specification	If a “+” separated list of hosts is specified in the Run Job request, and resources are specified with vnode names, e.g. NodeA:mem=100kb:ncpus=1+vnodeB:mem=100kb :ncpus=2, the Scheduler will apply the specified allocations and the select directive will be ignored. Any single resource specification will result in the job’s select directive being ignored.

A qrun command issued with the -H option may oversubscribe resources on a vnode, but it will not override the exclusive/shared allocation of a vnode. If a job is already running and the vnode is allocated to that prior job exclusively due to an explicit request of the job or due to the vnode's "sharing" attribute setting, an attempt to qrun an additional job on that vnode will result in the qrun being rejected and the job being left in the Queued state.

The qrun command can be used on a subjob or a range of subjobs, but not on a job array. When it is used on a range of subjobs, the non-running subjobs in that range are run.

7.16 The qmgr Command

The **`qmgr`** command is the Administrator interface to PBS, and is discussed in detail earlier in this book, in the section entitled “The qmgr Command” on page 8.

7.17 The qterm Command

The **`qterm`** command is used to shut down PBS, and is discussed in detail earlier in this book, in section 6.5.7 “Stopping PBS” on page 289.

7.18 The pbs_wish Command

The **`pbs_wish`** command is a version of TK Window Shell linked with a wrapped versions of the PBS Professional external API library. For usage see the `pbs_tclapi(3B)` manual page, and the **PBS Professional External Reference Specification.**

7.19 The qalter Command and Job Comments

Users tend to want to know what is happening to their job. PBS provides a special job attribute, `comment`, which is available to the operator, manager, or the Scheduler program. This attribute can be set to a string to pass information to the job owner. It might be used to display information about why the job is not being run or why a hold was placed on the job. Users are able to see this attribute, when set, by using the `-f and -s` options of the `qstat` command. (For details see “Displaying Job Comments” in the **PBS Professional User’s Guide**.) Operators and managers may use the `-W` option of the `qalter` command, for example

`qalter -W comment=”some text”` ***`job_id`***

The qalter command can be used on job array objects, but not on subjobs or ranges of subjobs. Note also that when used on a job array, the job ID must be enclosed in double quotes. See “qalter: Altering a Job Array” on page 215 of the **PBS Professional User’s Guide**.

Custom resources can be made invisible to users or unalterable by users via resource permission flags. See section 2.10.9 “Resource Flags for Resource Permissions” on page 71. If a user tries to alter a custom resource which has been made either invisible or unalterable, they will get an error message: “qalter: Cannot set attribute, read only or insufficient permission Resource_List.hps 173.mars”.

7.20 The pbs-report Command

The **`pbs-report`** command allows the PBS Administrator to generate a report of job statistics from the PBS accounting logfiles. Options are provided to filter the data reported based on start and end times for the report, as well as indicating specific data that should be reported. The available options are shown below, followed by sample output of the `pbs-report` command.

Important: The `pbs-report` command is not available on Windows.

Before first using `pbs-report`, the Administrator is advised to tune the **`pbs-report`** configuration to match the local site. This can be done by editing the file `PBS_EXEC/lib/pm/PBS.pm`.

Important: If job arrays are being used, the **`pbs-report`** command will produce errors including some about uninitialized variables. It will report on the job array object as well as on each subjob.

7.20.1 pbs-report Options

`--age -a seconds[:offset]`
Report age in seconds. If an offset is specified, the age range is taken from that offset backward in time, otherwise a zero offset is assumed. The time span is from (now - age - offset) to (now - offset). This option silently supersedes `--begin`, `--end`, and `--range`.

`--account account`
Limit results to those jobs with the specified account string. Multiple values may be concatenated with colons or specified with multiple instances of `--account`.

`--begin -b yyyymmdd[:hhmm[ss]]`
Report begin date and optional time (default: most recent log data).

`--count -c`	Display a numeric count of matching jobs. Currently only valid with `--cpumax` for use in monitoring rapidly-exiting jobs.
`--cpumax seconds`	Filter out any jobs which have more than the specified number of CPU seconds.
`--cpumin seconds`	Filter out any jobs which have less than the specified number of CPU seconds.
`--csv character`	Have the output be separated by the specified character. Currently only the "|" is supported. Character must be enclosed in double quotes.
`--dept -d department`	Limit results to those jobs whose owners are in the indicated department (default: any). This option only works in conjunction with an LDAP server which supplies department codes. See also the `--group` option. Multiple values may be concatenated with colons or specified with multiple instances of `--dept`.
`--end -e yyyymmdd[:hhmm[ss]]`	Report end date and optional time (default: most recent log data).
`--exit -x integer`	Limit results to jobs with the specified exit status (default: any).
`--explain-wait`	Print a reason for why jobs had to wait before running.
`--group -g group`	Limit results to the specified group name. Multiple values may be concatenated with colons or specified with multiple instances of `--group`.
`--help -h`	Prints all options and exits.

--host -m execution host

Limit results to the specified execution host. Multiple values may be concatenated with colons or specified with multiple instances of `--host`.

`--inclusive key`
: Limit results to jobs which had *both* start and end times in the range.

`--index -i key`
: Field on which to index the summary report (default: user). Valid values include: date, dept, host, package, queue, user.

`--man`
: Prints the manual page and exits.

`--negate -n option name`
: Logically negate the selected options; print all records *except* those that match the values for the selected criteria (default: unset; valid values: account, dept, exit, group, host, package, queue, user). Defaults cannot be negated; only options explicitly specified are negated. Multiple values may be concatenated with colons or specified with multiple instances of `--negate`.

`--package -p package`
: Limit results to the specified software package. Multiple values may be concatenated with colons or specified with multiple instances of `--package`. Valid values are can be seen by running a report with the `--index package` option. This option keys on custom resources requested at job submission time. Sites not using such custom resources will have all jobs reported under the catch-all *None* package with this option.

`--point yyyymmdd[:hhmm[ss]]`
: Print a report of all jobs which were actively running at the point in time specified. This option cannot be used with any other date or age option.

`--queue -q queue`
: Limit results to the specified queue. Multiple values may be concatenated with colons or specified with multiple instances of `--queue`. Note that if specific queues are defined via the `@QUEUES` line in

`PBS.pm`, then only those queues will be displayed. Leaving that parameter blank allows all queues to be displayed.

`--range -r date range` Provides a shorthand notation for current date ranges (default: all). Valid values are today, week, month, quarter, and year. This option silently supersedes `--begin` and `--end`, and is superseded by `--age`.

`--reslist` Include resource requests for all matching jobs. This option is mutually exclusive with `--verbose`.

`--sched -t` Generate a brief statistical analysis of Scheduler cycle times. No other data on jobs is reported.

`--sort -s field` Field by which to sort reports (default: user). Valid values are cpu, date, dept, host, jobs, package, queue, suspend (aka muda), wait, and wall.

To calculate muda:
1. Runtime is job end - job start.
2. If suspend is greater than 10 seconds, then suspend is runtime - walltime. Otherwise suspend is set to zero.
3. If cput is not zero, then muda is suspend/cput. Otherwise muda is zero.

`--time option` Used to indicate how time should be accounted. The default of `full` is to count the entire job's CPU and wall time in the report if the job ended during the report's date range. Optionally the `partial` option is used to cause only CPU and wall time during the report's date range to be counted.

`--user -u username` Limit results to the specified user name. Multiple values may be concatenated with colons or specified with multiple instances of `--user`.

`--verbose -v` Include attributes for all matching individual jobs (default: summary only). Job arrays will not be displayed, but subjobs will be displayed.

`--vsort field`	Field by which to sort the verbose output section reports (default: jobid). Valid values are cpu, date, exit, host, jobid, jobname, mem, name, package, queue, scratch, suspend, user, vmem, wall, wait. If neither `--verbose` nor `--reslist` is specified, `--vsort` is silently ignored. The scratch sort option is available only for resource reports (`--reslist`).
`--waitmax seconds`	Filter out any jobs which have more than the specified wait time in seconds.
`--waitmin seconds`	Filter out any jobs which have less than the specified wait time in seconds.
`--wallmax seconds`	Filter out any jobs which have more than the specified wall time in seconds.
`--wallmin seconds`	Filter out any jobs which have less than the specified wall time in seconds.
`--wall -w`	Use the walltime resource attribute rather than wall time calculated by subtracting the job start time from end time. The walltime resource attribute does *not* accumulate when a job is suspended for any reason, and thus may not accurately reflect the local interpretation of wall time.

Several options allow for filtering of which jobs to include. These options are as follows.

`--begin, --end, --range, --age, --point`	Each of these options allows the user to filter jobs by some range of dates or times. `--begin` and `--end` work from hard date limits. Omitting either will cause the report to contain all data to either the beginning or the end of the accounting data. Unbounded date reports may take several minutes to run, depending on the volume of work logged. `--range` is a short-hand way of selecting a prior date range and will supersede

--begin and --end. --age allows the user to select an arbitrary period going back a specified number of seconds from the time the report is run. --age will silently supersede all other date options. --point displays all jobs which were running at the specified point in time, and is incompatible with the other options. --point will produce an error if specified with any other date-related option.

--cpumax, --cpumin, --wallmax, --wallmin, --waitmax, --waitmin
: Each of these six options sets a filter which bounds the jobs on one of their three time attributes (CPU time, queue wait time, or wall time). A maximum value will cause any jobs with more than the specified amount to be ignored. A minimum value will cause any jobs with less than the specified amount to be ignored. All six options may be combined, though doing so will often restrict the filter such that no jobs can meet the requested criteria. Combine time filters for different time with caution.

--dept, --group, --user
: Each of these user-based filters allow the user to filter jobs based on who submitted them. --dept allows for integration with an LDAP server and will generate reports based on department codes as queried from that server. If no LDAP server is available, department-based filtering and sorting will not function. --group allows for filtering of jobs by primary group ownership of the submitting user, as defined by the operating system on which the PBS server runs. --user allows for explicit naming of users to be included. It is possible to specify a list of values for these filters, by providing a single colon-concatenated argument or using the option multiple times, each with a single value.

--account
: This option allows the user to filter jobs based on an arbitrary, user-specified job account string. The content and format of these strings is site-defined and unrestricted; it may be used by a custom job front-end which enforces permissible account strings, which are passed to qsub with qsub's -A option.

`--host`, `--exit`, `--package`, `--queue` — Each of these job-based filters allow the user to filter jobs based on some property of the job itself. `--host` allows for filtering of jobs based on the host on which the job was executed. `--exit` allows for filtering of jobs based on the job exit code. `--package` allows for filtering of jobs based on the software package used in the job. This option will only function when a package-specific custom resource is defined for the PBS server and requested by the jobs as they are submitted. `--queue` allows for filtering of jobs based on the queue in which the job finally executed. With the exception of `--exit`, it is possible to specify a list of values for these filters, by providing a single colon-concatenated argument or using the option multiple times, each with a single value.

`--negate` — The `--negate` option bears special mentioning. It allows for logical negation of one or more specified filters. Only the account, dept, exit, group, host, package, queue, and user filters may be negated. If a user is specified with `--user`, and the '`--negate user`' option is used, only jobs *not* belonging to that user will be included in the report. Multiple report filters may be negated by providing a single colon-concatenated argument or using `--negate` multiple times, each with a single value.

Several report types can be generated, each indexed and sorted according to the user's needs.

`--verbose` — This option generates a wide tabular output with detail for every job matching the filtering criteria. It can be used to generate output for import to a spreadsheet which can manipulate the data beyond what `pbs-report` currently provides. Verbose reports may be sorted on any field using the `--vsort` option. The default is to produce a summary report only.

`--reslist` This option generates a tabular output with detail on resources requested (*not* resources used) for every job matching the filtering criteria. Resource list reports may be sorted on any field using the `--vsort` option. The default is to produce a summary report only.

`--inclusive` Normal convention is to credit a job's entire run to the time at which it ends. So all date selections are bounds around the end time. This option allows a user to require that the job's start time also falls within the date range.

`--index` This option allows the user to select a field on which data in the summary should be grouped. The fields listed in the option description are mutually exclusive. Only one can be chosen, and will represent the left-most column of the summary report output. One value may be selected as an index while another is selected for sorting. However, since index values are mutually exclusive, the only sort options which may be used (other than the index itself) are account, cpu, jobs, suspend, wait, and wall. If no sort order is selected, the index is used as the sort key for the summary.

`--sort` This option allows the user to specify a field on which to sort the summary report. It operates independently of the sort field for verbose reports (see `--vsort`). See the description for `--index` for notes on how the two options interact.

`--vsort` This option allows the user to specify a field on which to sort the verbose report. It operates independently of the sort field for summary reports (see `--sort`).

`--time` This option allows the user to modify how time associated with a job is accounted. With *full*, all time is accounted for the job, and credited at the point when the job ended. For a job which ended a few seconds after the report range begins, this can cause significant overlap, which may boost results. During a sufficiently large time frame, this overlap effect is negligible and may be ignored. This value for `--time` should be used when generating monthly usage reports. With *partial*, any CPU or wall time accumulated prior to the beginning of the report is ignored. *partial* is intended to allow for more accurate calculation of overall cluster efficiency during short time spans during which a significant 'overlap' effect can skew results.

7.20.2 pbs-report Examples

This section explains several complex report queries to serve as examples for further experimentation. Note that some of options to `pbs-report` produce summary information of the resources requested by jobs (such as mem, vmem, ncpus, etc.). These resources are explained in Chapter 4 of the **PBS Professional User's Guide**.

Consider the following question: "This month, how much resources did every job which waited more than 10 minutes request?"

pbs-report --range month --waitmin 600 --reslist

This information might be valuable to determine if some simple resource additions (e.g. more memory or more disk) might increase overall throughput of the complex. At the bottom of the summary statistics, prior to the job set summary, is a statistical breakdown of the values in each column. For example:

```
               # of       Total       Total             Average
Date           jobs    CPU Time   Wall Time  Efcy.   Wait Time
----------    -----  ----------  ----------  -----  -----------
TOTAL          1900    10482613    17636290  0.594         1270
... individual rows indexed by date ...
Minimum           4        4715       13276  0.054          221
Maximum         162     1399894     2370006  1.782        49284
Mean             76      419304      705451  0.645         2943
Deviation        41      369271      616196  0.408         9606
Median           80      242685      436724  0.556          465
```

This summary should be read in column format. While the minimum number of jobs run in one day was 4 and the maximum 162, these values do ***not*** correlate to the 4715 and 1399894 CPU seconds listed as minimums and maximums.

In the Job Set Summary section, the values should be read in rows, as shown here:

```
                                              Standard
          Minimum     Maximum    Mean       Deviation    Median
          -------     -------    ------     --------     ------
CPU time      0        18730      343          812          0
Wall time     0       208190     8496        19711         93
Wait time     0       266822     4129         9018          3
```

These values represent aggregate statistical analysis for the entire set of jobs included in the report. The values in the prior summary represent values over the set of totals based on the summary index (e.g. Maximum and Minimum are the maximum and minimum totals for a given day/user/department, rather than an individual job. The job set summary represents an analysis of all individual jobs.

7.20.3 pbs-report Complex Monitoring

The `pbs-report` options `--count` and `--cpumax` are intended to allow an Administrator to periodically run this report to monitor for jobs which are exiting rapidly, representing a potential global error condition causing all jobs to fail. It is most useful in conjunction with --age, which allows a report to span an arbitrary number of seconds backward in time from the current moment. A typical set of options would be "`--count --cpumax 30 --age 21600`", which would show a total number of jobs which consumed less than 30 seconds of CPU time within the last six hours.

7.21 The `xpbs` Command (GUI) Admin Features

PBS currently provides two Graphical User Interfaces (GUIs): `xpbs` (intended primarily for users) and `xpbsmon` (intended for PBS operators and managers). Both are built using the Tool Control Language Toolkit (TCL/tk). The first section below discusses the user GUI, `xpbs`. The following section discusses `xpbsmon`.

7.21.1 xpbs GUI Configuration

xpbs provides a user-friendly point-and-click interface to the PBS commands. To run `xpbs` as a regular, non-privileged user, type:

```
xpbs
```

To run `xpbs` with the additional purpose of terminating PBS Servers, stopping and starting queues, running/rerunning jobs (as well as then run:

```
xpbs -admin
```

Important: See the manual page for xpbs, xpbs(1B), for a complete description of all xpbs functions.

Running `xpbs` will initialize the X resource database in order from the following sources:

1. The RESOURCE_MANAGER property on the root window (updated via `xrdb`) with settings usually defined in the `.Xdefaults` file
2. Preference settings defined by the system Administrator in the global `xpbsrc` file
3. User's `.xpbsrc` file-- this file defines various X resources like fonts, colors, list of PBS hosts to query, criteria for listing queues and jobs, and various view states.

The system Administrator can specify a global resources file to be read by the GUI if a personal `.xpbsrc` file is missing: `PBS_EXEC/lib/xpbs/xpbsrc`. Keep in mind that within an Xresources file (Tk only), later entries take precedence. For example, suppose in your `.xpbsrc` file, the following entries appear in order:

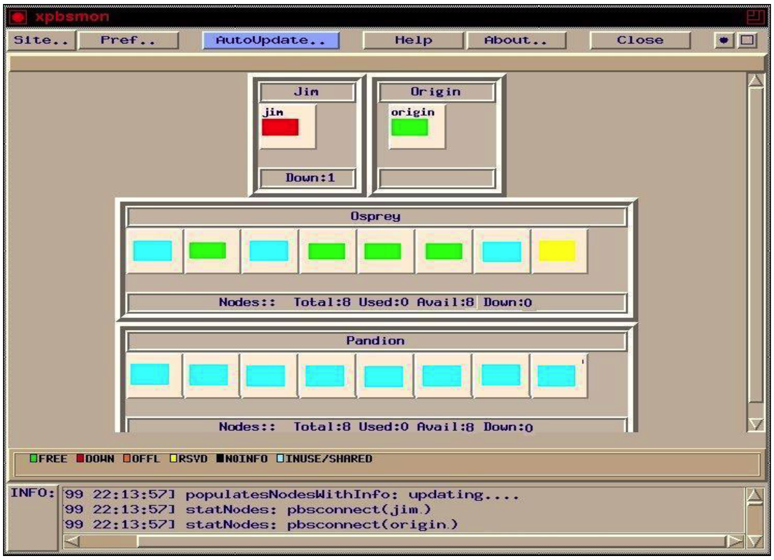
```
xpbsrc*backgroundColor: blue
*backgroundColor: green
```

The later entry "green" will take precedence even though the first one is more precise and longer matching. The things that can be set in the personal preferences file are fonts, colors, and favorite Server host(s) to query.

`xpbs` usage, command correlation, and further customization information is provided in the **PBS Professional User's Guide**, Chapter 5, "Using the xpbs GUI".

7.22 The xpbsmon GUI Command

xpbsmon is the vnode monitoring GUI for PBS. It is used for graphically displaying information about execution hosts in a PBS environment. Its view of a PBS environment consists of a list of sites where each site runs one or more Servers, and each Server runs jobs on one or more execution hosts (vnodes).



The system Administrator needs to define the site's information in a global X resources file, *PBS_EXEC/lib/*`xpbsmon/xpbsmonrc` which is read by the GUI if a personal `.xpbsmonrc` file is missing. A default `xpbsmonrc` file usually would have been created already during installation, defining (under *sitesInfo resource) a default site name, list of Servers that run on a site, set of vnodes (or execution hosts) where jobs on a particular Server run, and the list of queries that are communicated to each vnode's pbs_mom. If vnode queries have been specified, the host where xpbsmon is running must have been given explicit permission by the pbs_mom to post queries to it. This is done by including a $restricted entry in the MOM's config file. It is not recommended to manually update the *sitesInfo value in the xpbsmonrc file as its syntax is quite cumbersome. The recommended procedure is to bring up xpbsmon, click on "Pref.." button, manipulate the widgets in the Sites, Server, and Query Table dialog boxes, then click "Close" button and save the settings to a .xpbsmonrc file. Then copy this file over to the *PBS_EXEC*/lib/xpbsmon/ directory.

7.23 The pbskill Command

Under Microsoft Windows XP, PBS includes the **pbskill** utility to terminate any job related tasks or processes. DOS/Windows prompt usage for the pbskill utility is:

pbskill processID1 [[processID2] [processID3] ...]

Note that Under Windows, if the pbskill command is used to terminate the MOM service, it may leave job processes running, which if present, will prevent a restart of MOM (a "network is busy" message will be reported). This can be resolved by manually killing the errant job processes via the Windows task manager.

Chapter 8

Example Configurations

Up to this point in this manual, we have seen many examples of how to configure the individual PBS components, set limits, and otherwise tune a PBS installation. Those examples were used to illustrate specific points or configuration options. This chapter pulls these various examples together into configuration-specific scenarios which will hopefully clarify any remaining configuration questions. Several configuration models are discussed, followed by several complex examples of specific features.

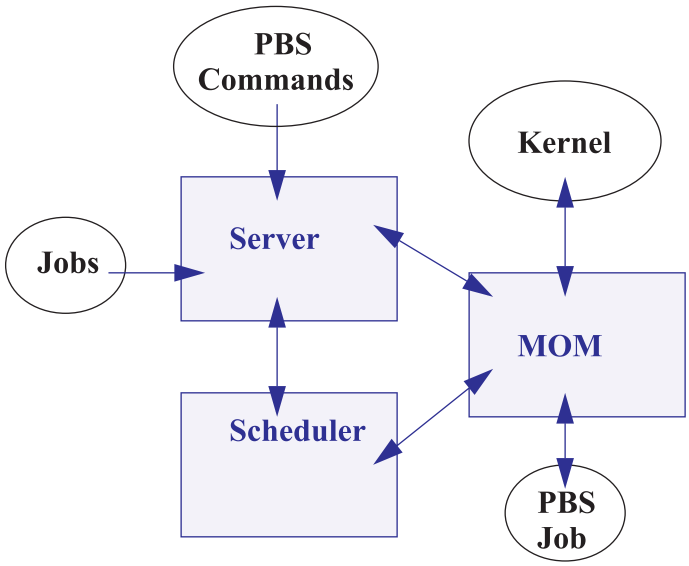
Single Vnode System
Single Vnode System with Separate PBS Server
Multi-vnode complex
Complex Multi-level Route Queues (including group ACLs)
Multiple User ACLs

For each of these possible configuration models, the following information is provided:

General description for the configuration model
Type of system for which the model is well suited
Contents of Server nodes file
Any required Server configuration
Any required MOM configuration
Any required Scheduler configuration

8.1 Single Vnode System

Running PBS on a single vnode/host as a standalone system is the least complex configuration. This model is most applicable to sites who have a single large Server system, a single SMP system (e.g. an SGI Origin server), or even a vector supercomputer. In this model, all three PBS components run on the same host, which is the same host on which jobs will be executed, as shown in the figure below.



All components on a single host.

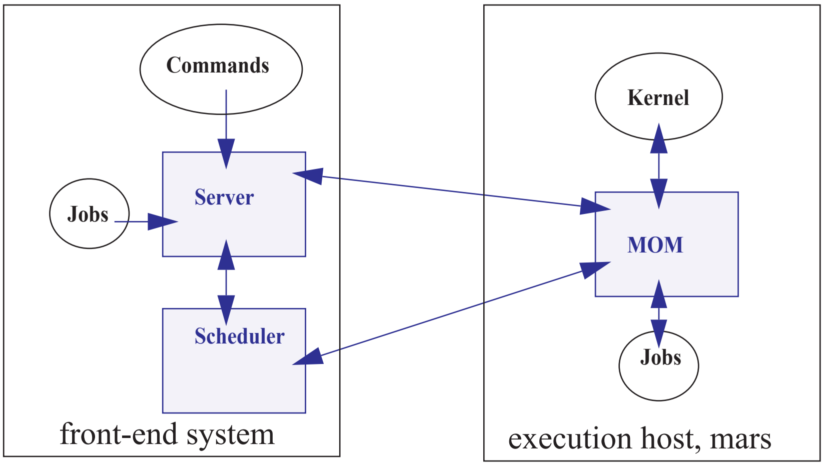
For this example, let's assume we have a 32-CPU server machine named "mars". We want users to log into mars and jobs will be run via PBS on mars.

In this configuration, the server's default `nodes` file (which should contain the name of the host on which the Server was installed) is sufficient. Our example `nodes` file would contain only one entry: `mars`

The default MOM and Scheduler `config` files, as well as the default queue/Server limits are also sufficient in order to run jobs. No changes are required from the default configuration, however, you may wish to customize PBS to your site.

8.2 Separate Server and Execution Host

A variation on the model presented above would be to provide a "front-end" system that ran the PBS Server and Scheduler, and from which users submitted their jobs. Only the MOM would run on our execution server, mars. This model is recommended when the user load would otherwise interfere with the computational load on the Server.



In this case, the PBS `server_priv/nodes` file would contain the name of our execution server mars, but this may not be what was written to the file during installation, depending on which options were selected. It is possible the hostname of the machine on which the Server was installed was added to the file, in which case you would need to use `qmgr(1B)` to manipulate the contents to contain one vnode: `mars`. If the default scheduling policy, based on available CPUs and memory, meets your requirements, then no changes are required in either the MOM or Scheduler configuration files.

However, if you wish the execution host (mars) to be scheduled based on load average, the following changes are needed. Edit MOM's `mom_priv/config` file so that it contains the target and maximum load averages, e.g.:

```
$ideal_load 30
$max_load 32
```

In the Scheduler `sched_priv/sched_config` file, the following options would need to be set:

```
load_balancing: true all
```

8.3 Multiple Execution Hosts

The multi-vnode complex model is a very common configuration for PBS. In this model, there is typically a front-end system as we saw in the previous example, with a number of back-end execution hosts. The PBS Server and Scheduler are typically run on the front-end system, and a MOM is run on each of the execution hosts, as shown in the diagram to the right.

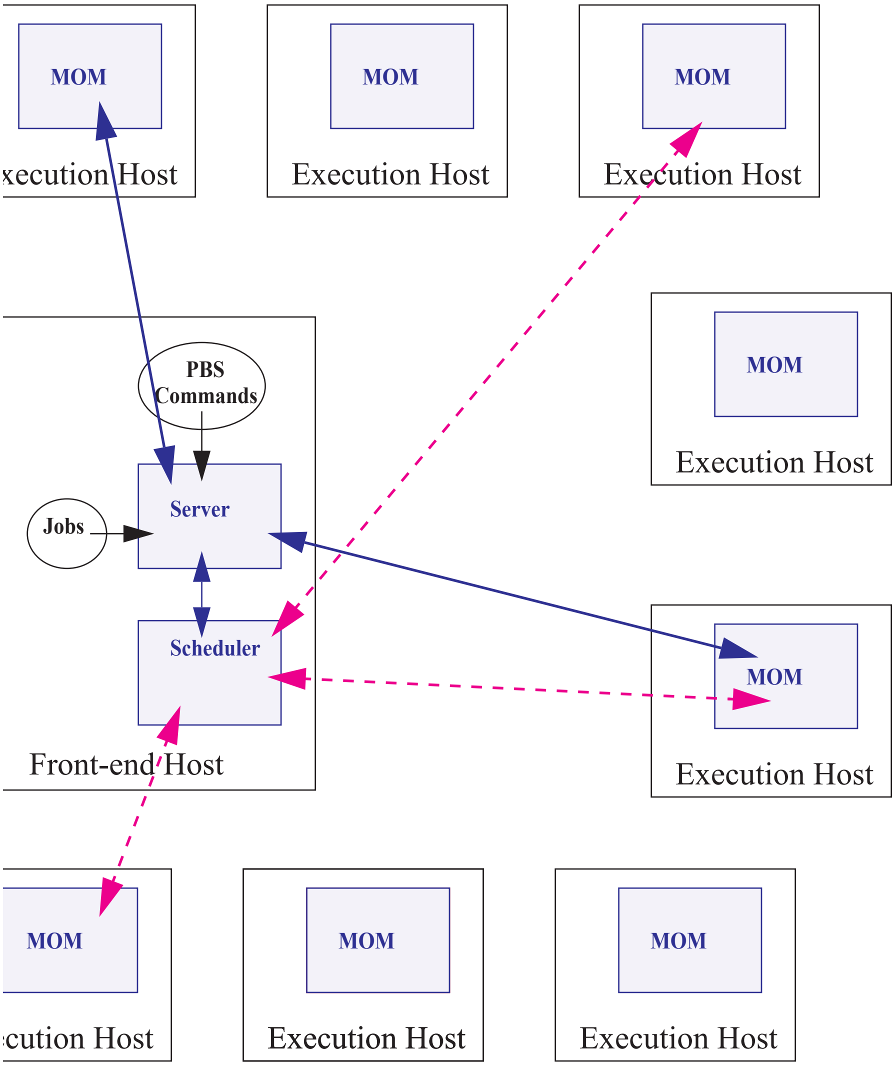
In this model, the server's `nodes` file will need to contain the list of all the vnodes in the complex.

The MOM `config` file on each vnode will need two static resources added, to specify the target load for each vnode. If we assume each of the vnodes in our "planets" cluster is a 32-processor system, then the following example shows what might be desirable ideal and maximum load values to add to the MOM `config` files:

```
$ideal_load 30
$max_load 32
```

Furthermore, suppose we want the Scheduler to load balance the workload across the available vnodes, making sure not to run two jobs in a row on the same vnode (round robin vnode scheduling). We accomplish this by editing the Scheduler configuration file and enabling load balancing:

load_balancing: **true all**
smp_cluster_dist: **round_robin**



This diagram illustrates a multi-vnode complex configuration wherein the Scheduler and Server communicate with the MOMs on the execution hosts. Jobs are submitted to the Server, scheduled for execution by the Scheduler, and then transferred to a MOM when it's time to be run. MOM periodically sends status information back to the Server, and answers resource requests from the Scheduler.

8.4 Complex Multi-level Route Queues

There are times when a site may wish to create a series of route queues in order to filter jobs, based on specific resources, or possibly to different destinations. For this example, consider a site that has two large Server systems, and a Linux cluster. The Administrator wants to configure route queues such that everyone submits jobs to a single queue, but the jobs get routed based on (1) requested architecture and (2) individual group IDs. In other words, users request the architecture they want, and PBS finds the right queue for them. Only groups “math”, “chemistry”, and “physics” are permitted to use either server systems; while anyone can use the cluster. Lastly, the jobs coming into the cluster should be divided into three separate queues for long, short, and normal jobs. But the “long” queue was created for the astronomy department, so only members of that group should be permitted into that queue. Given these requirements, let's look at how we would set up such a collection of route queues. (Note that this is only one way to accomplish this task. There are various other ways too.)

First we create a queue to which everyone will submit their jobs. Let's call it “submit”. It will need to be a route queue with three destinations, as shown:

```
qmgr
Qmgr: create queue submit
Qmgr: set queue submit queue_type = Route
Qmgr: set queue submit route_destinations = server_1
Qmgr: set queue submit route_destinations += server_2
Qmgr: set queue submit route_destinations += cluster
Qmgr: set queue submit enabled = True
Qmgr: set queue submit started = True
```

Now we need to create the destination queues. (Notice in the above example, we have already decided what to call the three destinations: `server_1`, `server_2`, `cluster`.) First we create the `server_1` queue, complete with a group ACL, and a specific architecture limit.

Qmgr: **create queue server_1**
Qmgr: **set queue server_1 queue_type = Execution**
Qmgr: **set queue server_1 from_route_only = True**
Qmgr: **set queue server_1 resources_max.arch = linux**
Qmgr: **set queue server_1 resources_min.arch = linux**
Qmgr: **set queue server_1 acl_group_enable = True**
Qmgr: **set queue server_1 acl_groups = math**
Qmgr: **set queue server_1 acl_groups += chemistry**
Qmgr: **set queue server_1 acl_groups += physics**
Qmgr: **set queue server_1 enabled = True**
Qmgr: **set queue server_1 started = True**

Next we create the queues for `server_2` and `cluster`. Note that the `server_2` queue is very similar to the `server_1` queue, only the architecture differs. Also notice that the `cluster` queue is another route queue, with multiple destinations.

Qmgr: **create queue server_2**
Qmgr: **set queue server_2 queue_type = Execution**
Qmgr: **set queue server_2 from_route_only = True**
Qmgr: **set queue server_2 resources_max.arch = sv2**
Qmgr: **set queue server_2 resources_min.arch = sv2**
Qmgr: **set queue server_2 acl_group_enable = True**
Qmgr: **set queue server_2 acl_groups = math**
Qmgr: **set queue server_2 acl_groups += chemistry**
Qmgr: **set queue server_2 acl_groups += physics**
Qmgr: **set queue server_2 enabled = True**
Qmgr: **set queue server_2 started = True**
Qmgr: **create queue cluster**
Qmgr: **set queue cluster queue_type = Route**
Qmgr: **set queue cluster from_route_only = True**
Qmgr: **set queue cluster resources_max.arch = linux**
Qmgr: **set queue cluster resources_min.arch = linux**
Qmgr: **set queue cluster route_destinations = long**
Qmgr: **set queue cluster route_destinations += short**
Qmgr: **set queue cluster route_destinations += medium**
Qmgr: **set queue cluster enabled = True**
Qmgr: **set queue cluster started = True**

In the cluster queue above, you will notice the particular order of the three destination queues (`long`, `short`, `medium`). PBS will attempt to route a job into the destination queues in the order specified. Thus, we want PBS to first try the `long` queue (which will have an ACL on it), then the `short` queue (with its short time limits). Thus any jobs that had not been routed into any other queues (server or cluster) will end up in the `medium` cluster queue. Now to create the remaining queues.

Qmgr: **create queue long**
Qmgr: **set queue long queue_type = Execution**
Qmgr: **set queue long from_route_only = True**
Qmgr: **set queue long resources_max.cput = 20:00:00**
Qmgr: **set queue long resources_max.walltime = 20:00:00**
Qmgr: **set queue long resources_min.cput = 02:00:00**
Qmgr: **set queue long resources_min.walltime = 03:00:00**
Qmgr: **set queue long acl_group_enable = True**
Qmgr: **set queue long acl_groups = astrology**
Qmgr: **set queue long enabled = True**
Qmgr: **set queue long started = True**

Qmgr: **create queue short**
Qmgr: **set queue short queue_type = Execution**
Qmgr: **set queue short from_route_only = True**
Qmgr: **set queue short resources_max.cput = 01:00:00**
Qmgr: **set queue short resources_max.walltime = 01:00:00**
Qmgr: **set queue short enabled = True**
Qmgr: **set queue short started = True**
Qmgr: **create queue medium**
Qmgr: **set queue medium queue_type = Execution**
Qmgr: **set queue medium from_route_only = True**
Qmgr: **set queue medium enabled = True**
Qmgr: **set queue medium started = True**
Qmgr: **set server default_queue = submit**

Notice that the `long` and `short` queues have time limits specified. This will ensure that jobs of certain sizes will enter (or be prevented from entering) these queues. The last queue, `medium`, has no limits, thus it will be able to accept any job that is not routed into any other queue.

Lastly, note the last line in the example above, which specified that the default queue is the new `submit` queue. This way users will simply submit their jobs with the resource and architecture requests, without specifying a queue, and PBS will route the job into the correct location. For example, if a user submitted a job with the following syntax, the job would be routed into the `server_2` queue:

```
qsub -l select=arch=sv2:ncpus=4 testjob
```

8.5 External Software License Management

PBS Professional can be configured to schedule jobs based on externally-controlled licensed software. A detailed example is provided in section 5.7.4 “Example of Floating, Externally-managed License with Features” on page 260.

8.6 Multiple User ACL Example

A site may have a need to restrict individual users to particular queues. In the previous example we set up queues with group-based ACLs, in this example we show user-based ACLs. Say a site has two different groups of users, and wants to limit them to two separate queues (perhaps with different resource limits). The following example illustrates this.

```
Qmgr: create queue structure
Qmgr: set queue structure queue_type = Execu-
tion
Qmgr: set queue structure acl_user_enable =
True
Qmgr: set queue structure acl_users = curly
Qmgr: set queue structure acl_users += jerry
Qmgr: set queue structure acl_users += larry
Qmgr: set queue structure acl_users += moe
Qmgr: set queue structure acl_users += tom
Qmgr: set queue structure resources_max.nodes
```

```
= 48
Qmgr: set queue structure enabled = True
Qmgr: set queue structure started = True
Qmgr:
Qmgr: create queue engine
Qmgr: set queue engine queue_type = Execution
Qmgr: set queue engine acl_user_enable = True
Qmgr: set queue engine acl_users = bill
Qmgr: set queue engine acl_users += bobby
Qmgr: set queue engine acl_users += chris
Qmgr: set queue engine acl_users += jim
Qmgr: set queue engine acl_users += mike
Qmgr: set queue engine acl_users += rob
Qmgr: set queue engine acl_users += scott
Qmgr: set queue engine resources_max.nodes =
12
Qmgr: set queue engine resources_max.wall-
time=04:00:00
Qmgr: set queue engine enabled = True
Qmgr: set queue engine started = True
Qmgr:
```

Chapter 9

Problem Solving

The following is a list of common problems and recommended solutions. Additional information is always available online at the PBS website, www.pbspro.com/UserArea. The last section in this chapter gives important information on how to get additional assistance from the PBS Support staff.

9.1 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.

9.2 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. For details on usage of the `pbs_probe` command see section 7.4 “The pbs_probe Command” on page 369.

9.3 Job Exit Codes

The exit value of a job may fall in one of three ranges: X < 0, 0 <=X < 128, X >=128.

X < 0:
This is a PBS special return value indicating that the job could not be executed. These negative values are listed in the table below.

0 <= X < 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).

X >= 128 (or 256 depending on the system)
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).

Table 1:

	Name	Description
0	JOB_EXEC_OK	job exec successful
-1	JOB_EXEC_FAIL1	Job exec failed, before files, no retry
-2	JOB_EXEC_FAIL2	Job exec failed, after files, no retry
-3	JOB_EXEC_RETRY	Job execution failed, do retry

Table 1:

	Name	Description
-4	JOB_EXEC_INITABT	Job aborted on MOM initialization
-5	JOB_EXEC_INITRST	Job aborted on MOM init, chkpt, no migrate
-6	JOB_EXEC_INITRMG	Job aborted on MOM init, chkpt, ok migrate
-7	JOB_EXEC_ BADRESRT	Job restart failed
-8	JOB_EXEC_GLOBUS_ INIT_RETRY	Init. globus job failed. do retry
-9	JOB_EXEC_GLOBUS_ INIT_FAIL	Init. globus job failed. no retry
-10	JOB_EXEC_FAILUID	invalid uid/gid for job
-11	JOB_EXEC_RERUN	Job rerun
-12	JOB_EXEC_CHKP	Job was checkpointed and killed
-13	JOB_EXEC_FAIL_ PASSWORD	Job failed due to a bad password
-14	JOB_EXEC_RERUN_ ON_SIS_FAIL	Job was requeued (if rerunnable) or deleted (if not) due to a communication failure between Mother Superior and a Sister

The PBS Server logs and accounting logs record the exit status of jobs. Zero or positive exit status is the status of the top level shell. The positive exit status values indicate which signal killed the job. Depending on the system, values greater than 128 (or on some systems 256; see wait(2) or waitpid(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.

9.4 Common Errors

9.4.1 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 no error and a non-zero status, either the Server is not running or was installed with a different encryption system than was `pbs_iff`.

9.4.2 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
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.

9.4.3 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 (see "node_fail_requeue" on page 27), or if you wish to act immediately, you can manually force the requeueing and/or rerunning of the job.

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
```

9.4.4 File Stagein Failure

When stagein fails, the job is placed in a 30-minute wait to allow the user time to fix the problem. Typically this is a missing file or a network outage. Email is sent to the job owner when the problem is detected. Once the problem has been resolved, the job owner or the Operator may remove the wait by resetting the time after which the job is eligible to be run via the `-a` option to **`qalter`**. The server will update the job's comment with information about why the job was put in the wait state. The job's exec_host string is cleared so that it can run on any vnode(s) once it is eligible.

9.4.5 File Stageout Failure

When stageout encounters an error, there are three retries. PBS waits 1 second and tries again, then waits 11 seconds and tries a third time, then finally waits another 21 seconds and tries a fourth time. PBS sends the job's owner email if the stageout is unsuccessful. For each attempt, if PBS is using scp and that doesn't work, PBS will then try rcp.

9.4.6 Non Delivery of Output

If the output of a job cannot be delivered to the user, it is saved in a special directory:
PBS_HOME`/undelivered` and mail is sent to the user. The typical causes of non-delivery are:

1 The destination host is not trusted and the user does not have a `.rhosts` file.

2 An improper path was specified.

3 A directory in the specified destination path is not writable.

4 The user's `.cshrc` on the destination host generates output when executed.

5 The path specified by **`PBS_SCP`** in `pbs.conf` is incorrect.

6 The *PBS_HOME*`/spool` directory on the execution host does not have

the correct permissions. This directory must have mode 1777 `drwxr-wxrwxt` (on UNIX) or “Full Control” for “Everyone” (on Windows).

See also the “Delivery of Output Files” section of the **PBS Professional User’s Guide**.

9.4.7 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.

9.4.8 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.

9.4.9 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**.

9.4.10 SuSE 9.1 with mpirun and ssh

Use “`ssh -n`” instead of “`ssh`”.

9.4.11 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.

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.

9.5 Common Errors on Windows

This section discusses errors often encountered under Windows.

9.5.1 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

9.5.2 MOMs Won’t Start

In a domained environment, if the pbsadmin account is a member of any group besides "Domain Users", the install program will fail to add pbsadmin to the local Administrators group on the install host. Make sure that pbsadmin is a member of only one group, "Domain Users" in a domained environment.

9.5.3 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.

9.5.4 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 also section 6.8.5 "User Authorization" on page 299.

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.

9.5.5 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.

9.5.6 Windows: Server Reports Error 10054

If the Server reports error 10054 `rp_request()`, this indicates that another process, probably `pbs_sched`, `pbs_mom`, or `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:

```
pbskill process-id
```

9.5.7 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 `pbs_environment`, `server_priv/nodes`, `mom_priv/config`, then correct the permission by running:

```
pbs_mkdirs server
pbs_mkdirs mom
pbs_mkdirs sched
```

9.5.8 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 `pbs.conf` global configuration file. If this is the case, PBS will report the following message:

```
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)
```

To correct this problem, set `PBS_CONF_FILE` to point `pbs.conf` to the right path. Normally, during PBS Windows installation, this would be set in system `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.

9.5.9 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
```

9.5.10 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.

9.6 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.

The PBS Professional support team can also be reached directly via email and phone (contact information on the inside front cover of this manual).

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.)

9.7 Troubleshooting PBS Licenses

9.7.1 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_file_location=<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_file_location=<X>). No external license server will be contacted”

9.7.2 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>”

9.7.3 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)“

9.7.4 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.

9.7.5 Insufficient Minimum Licenses

If the PBS server cannot get the number of licenses specified in `pbs_license_min` from the FLEX 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."

9.7.6 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".

9.7.7 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>"

Appendix A: Error Codes

The following table lists all the PBS error codes, their textual names, and a description of each.

Table 1:

Error Name	Error Code	Description
PBSE_NONE	0	No error
PBSE_UNKJOBID	15001	Unknown Job Identifier
PBSE_NOATTR	15002	Undefined Attribute

Table 1:

Error Name	Error Code	Description
PBSE_ATTRRO	15003	Attempt to set READ ONLY attribute
PBSE_IVALREQ	15004	Invalid request
PBSE_UNKREQ	15005	Unknown batch request
PBSE_TOOMANY	15006	Too many submit retries
PBSE_PERM	15007	No permission
PBSE_BADHOST	15008	Access from host not allowed
PBSE_JOBEXIST	15009	Job already exists
PBSE_SYSTEM	15010	System error occurred
PBSE_INTERNAL	15011	Internal Server error occurred
PBSE_REGROUTE	15012	Parent job of dependent in route queue
PBSE_UNKSIG	15013	Unknown signal name
PBSE_BADATVAL	15014	Bad attribute value
PBSE_MODATRRUN	15015	Cannot modify attrib in run state
PBSE_BADSTATE	15016	Request invalid for job state
PBSE_UNKQUE	15018	Unknown queue name
PBSE_BADCRED	15019	Invalid Credential in request
PBSE_EXPIRED	15020	Expired Credential in request
PBSE_QUNOENB	15021	Queue not enabled
PBSE_QACESS	15022	No access permission for queue
PBSE_BADUSER	15023	Missing userID, username, or GID.

Appendix A: Error Codes

Table 1:

Error Name	Error Code	Description
PBSE_HOPCOUNT	15024	Max hop count exceeded
PBSE_QUEEXIST	15025	Queue already exists
PBSE_ATTRTYPE	15026	Incompatible queue attribute type
PBSE_OBJBUSY	15027	Object Busy
PBSE_QUENBIG	15028	Queue name too long
PBSE_NOSUP	15029	Feature/function not supported
PBSE_QUENOEN	15030	Can't enable queue, lacking definition
PBSE_PROTOCOL	15031	Protocol (ASN.1) error
PBSE_BADATLST	15032	Bad attribute list structure
PBSE_NOCONNECTS	15033	No free connections
PBSE_NOSERVER	15034	No Server to connect to
PBSE_UNKRESC	15035	Unknown resource
PBSE_EXCQRESC	15036	Job exceeds Queue resource limits
PBSE_QUENODFLT	15037	No Default Queue Defined
PBSE_NORERUN	15038	Job Not Rerunnable
PBSE_ROUTEREJ	15039	Route rejected by all destina-tions
PBSE_ROUTEEXPD	15040	Time in Route Queue Expired
PBSE_MOMREJECT	15041	Request to MOM failed
PBSE_BADSCRIPT	15042	(qsub) Cannot access script file
PBSE_STAGEIN	15043	Stage In of files failed

Table 1:

Error Name	Error Code	Description
PBSE_RESCUNAV	15044	Resources temporarily unavailable
PBSE_BADGRP	15045	Bad Group specified
PBSE_MAXQUED	15046	Max number of jobs in queue
PBSE_CKPBSY	15047	Checkpoint Busy, may be retries
PBSE_EXLIMIT	15048	Limit exceeds allowable
PBSE_BADACCT	15049	Bad Account attribute value
PBSE_ALRDYEXIT	15050	Job already in exit state
PBSE_NOCOPYFILE	15051	Job files not copied
PBSE_CLEANEDOUT	15052	Unknown job id after clean init
PBSE_NOSYNCMSTR	15053	No Master in Sync Set
PBSE_BADDEPEND	15054	Invalid dependency
PBSE_DUPLIST	15055	Duplicate entry in List
PBSE_DISPROTO	15056	Bad DIS based Request Proto-col
PBSE_EXECTHERE	15057	Cannot execute there
PBSE_SISREJECT	15058	Sister rejected
PBSE_SISCOMM	15059	Sister could not communicate
PBSE_SVRDOWN	15060	Request rejected -server shut-ting down
PBSE_CKPSHORT	15061	Not all tasks could checkpoint
PBSE_UNKNODE	15062	Named vnode is not in the list

Table 1:

Error Name	Error Code	Description
PBSE_UNKNODEATR	15063	Vnode attribute not recognized
PBSE_NONODES	15064	Server has no vnode list
PBSE_NODENBIG	15065	Node name is too big
PBSE_NODEEXIST	15066	Node name already exists
PBSE_BADNDATVAL	15067	Bad vnode attribute value
PBSE_MUTUALEX	15068	State values are mutually exclusive
PBSE_GMODERR	15069	Error(s) during global mod of vnodes
PBSE_NORELYMOM	15070	Could not contact MOM
PBSE_RESV_NO_WALLTIME	15075	Reservation lacking walltime
Reserved	15076	Not used.
PBSE_TOOLATE	15077	Reservation submitted with a start time that has already passed
PBSE_IRESVE	15078	Internal reservation-system error
PBSE_UNKRESVTYPE	15079	Unknown reservation type
PBSE_RESVEXIST	15080	Reservation already exists
PBSE_resvFail	15081	Reservation failed
PBSE_genBatchReq	15082	Batch request generation failed
PBSE_mgrBatchReq	15083	qmgr batch request failed
PBSE_UNKRESVID	15084	Unknown reservation ID

Table 1:

Error Name	Error Code	Description
PBSE_delProgress	15085	Delete already in progress
PBSE_BADTSPEC	15086	Bad time specification(s)
PBSE_RESVMSG	15087	So reply_text can return a msg
PBSE_NOTRESV	15088	Not a reservation
PBSE_BADNODESPEC	15089	Node(s) specification error
PBSE_LICENSECPU	15090	Licensed CPUs exceeded
PBSE_LICENSEINV	15091	License is invalid
PBSE_RESVAUTH_H	15092	Host not authorized to make AR
PBSE_RESVAUTH_G	15093	Group not authorized to make AR
PBSE_RESVAUTH_U	15094	User not authorized to make AR
PBSE_R_UID	15095	Bad effective UID for reservation
PBSE_R_GID	15096	Bad effective GID for reservation
PBSE_IBMSPSWITCH	15097	IBM SP Switch error
PBSE_LICENSEUNAV	15098	Floating License unavailable
	15099	UNUSED
PBSE_RESCNOTSTR	15100	Resource is not of type string
PBSE_SSIGNON_UNSET_REJECT	15101	rejected if SVR_ssignon_enable not set
PBSE_SSIGNON_SET_REJECT	15102	rejected if SVR_ssignon_enable set

Table 1:

Error Name	Error Code	Description
PBSE_SSIGNON_BAD_ TRANSITION1	15103	bad attempt: true to false
PBSE_SSIGNON_BAD_ TRANSITION2	15104	bad attempt: false to true
PBSE_SSIGNON_ NOCONNECT_DEST	15105	couldn't connect to destination host during a user migration request
PBSE_SSIGNON_NO_ PASSWORD	15106	no per-user/per-server password
Resource monitor specific error codes		
PBSE_RMUNKNOWN	15201	Resource unknown
PBSE_RMBADPARAM	15202	Parameter could not be used
PBSE_RMNOPARAM	15203	A needed parameter did not exist
PBSE_RMEXIST	15204	Something specified didn't exist
PBSE_RMSYSTEM	15205	A system error occurred
PBSE_RMPART	15206	Only part of reservation made

Appendix B: Request Codes

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. The following table lists all the PBS type codes and the corresponding request of each.

Table 1:

0	PBS_BATCH_Connect
1	PBS_BATCH_QueueJob
2	UNUSED
3	PBS_BATCH_jobscript
4	PBS_BATCH_RdytoCommit

Table 1:

5	PBS_BATCH_Commit
6	PBS_BATCH_DeleteJob
7	PBS_BATCH_HoldJob
8	PBS_BATCH_LocateJob
9	PBS_BATCH_Manager
10	PBS_BATCH_MessJob
11	PBS_BATCH_ModifyJob
12	PBS_BATCH_MoveJob
13	PBS_BATCH_ReleaseJob
14	PBS_BATCH_Rerun
15	PBS_BATCH_RunJob
16	PBS_BATCH_SelectJobs
17	PBS_BATCH_Shutdown
18	PBS_BATCH_SignalJob
19	PBS_BATCH_StatusJob
20	PBS_BATCH_StatusQue
21	PBS_BATCH_StatusSvr
22	PBS_BATCH_TrackJob
23	PBS_BATCH_AsyrunJob
24	PBS_BATCH_Rescq
25	PBS_BATCH_ReserveResc
26	PBS_BATCH_ReleaseResc
27	PBS_BATCH_FailOver
48	PBS_BATCH_StageIn
49	PBS_BATCH_AuthenUser

Appendix B: Request Codes

Table 1:

50	PBS_BATCH_OrderJob
51	PBS_BATCH_SelStat
52	PBS_BATCH_RegistDep
54	PBS_BATCH_CopyFiles
55	PBS_BATCH_DelFiles
56	PBS_BATCH_JobObit
57	PBS_BATCH_MvJobFile
58	PBS_BATCH_StatusNode
59	PBS_BATCH_Disconnect
60	UNUSED
61	UNUSED
62	PBS_BATCH_JobCred
63	PBS_BATCH_CopyFiles_Cred
64	PBS_BATCH_DelFiles_Cred
65	PBS_BATCH_GSS_Context
66	UNUSED
67	UNUSED
68	UNUSED
69	UNUSED
70	PBS_BATCH_SubmitResv
71	PBS_BATCH_StatusResv
72	PBS_BATCH_DeleteResv
73	PBS_BATCH_UserCred
74	PBS_BATCH_UserMigrate

Appendix B: Request Codes

Appendix C: File Listing

The following table lists all the PBS files and directories; owner and permissions are specific to UNIX systems.

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_HOME	root	drwxr-xr-x	4096
PBS_HOME/pbs_environment	root	-rw-r--r--	0
PBS_HOME/server_logs	root	drwxr-xr-x	4096
PBS_HOME/spool	root	drwxr-wxrwt	4096
PBS_HOME/server_priv	root	drwxr-x---	4096

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_HOME/server_priv/accounting	root	drwxr-xr-x	4096
PBS_HOME/server_priv/acl_groups	root	drwxr-x---	4096
PBS_HOME/server_priv/acl_hosts	root	drwxr-x---	4096
PBS_HOME/server_priv/acl_svr	root	drwxr-x---	4096
PBS_HOME/server_priv/acl_svr/managers	root	-rw-------	13
PBS_HOME/server_priv/acl_users	root	drwxr-x---	4096
PBS_HOME/server_priv/jobs	root	drwxr-x---	4096
PBS_HOME/server_priv/queues	root	drwxr-x---	4096
PBS_HOME/server_priv/queues/workq	root	-rw-------	303
PBS_HOME/server_priv/queues/newqueue	root	-rw-------	303
PBS_HOME/server_priv/resvs	root	drwxr-x---	4096
PBS_HOME/server_priv/nodes	root	-rw-r--r--	59
PBS_HOME/server_priv/server.lock	root	-rw-------	4
PBS_HOME/server_priv/tracking	root	-rw-------	0
PBS_HOME/server_priv/serverdb	root	-rw-------	876
PBS_HOME/server_priv/license_file	root	-rw-r--r--	34
PBS_HOME/aux	root	drwxr-xr-x	4096
PBS_HOME/checkpoint	root	drwx------	4096
PBS_HOME/mom_logs	root	drwxr-xr-x	4096
PBS_HOME/mom_priv	root	drwxr-x--x	4096
PBS_HOME/mom_priv/jobs	root	drwxr-x--x	4096

Appendix C: File Listing

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_HOME/mom_priv/config	root	-rw-r--r--	18
PBS_HOME/mom_priv/mom.lock	root	-rw-r--r--	4
PBS_HOME/undelivered	root	drwxr-wxrwt	4096
PBS_HOME/sched_logs	root	drwxr-xr-x	4096
PBS_HOME/sched_priv	root	drwxr-x---	4096
PBS_HOME/sched_priv/dedicated_time	root	-rw-r--r--	557
PBS_HOME/sched_priv/holidays	root	-rw-r--r--	1228
PBS_HOME/sched_priv/sched_config	root	-rw-r--r--	6370
PBS_HOME/sched_priv/resource_group	root	-rw-r--r--	0
PBS_HOME/sched_priv/sched.lock	root	-rw-r--r--	4
PBS_HOME/sched_priv/sched_out	root	-rw-r--r--	0
PBS_EXEC/	root	drwxr-xr-x	4096
PBS_EXEC/bin	root	drwxr-xr-x	4096
PBS_EXEC/bin/nqs2pbs	root	-rwxr-xr-x	16062
PBS_EXEC/bin/pbs_hostn	root	-rwxr-xr-x	35493
PBS_EXEC/bin/pbs_rdel	root	-rwxr-xr-x	151973
PBS_EXEC/bin/pbs_rstat	root	-rwxr-xr-x	156884
PBS_EXEC/bin/pbs_rsub	root	-rwxr-xr-x	167446
PBS_EXEC/bin/pbs_tclsh	root	-rwxr-xr-x	857552
PBS_EXEC/bin/pbs_wish	root	-rwxr-xr-x	1592236
PBS_EXEC/bin/pbsdsh	root	-rwxr-xr-x	111837

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/bin/pbsnodes	root	-rwxr-xr-x	153004
PBS_EXEC/bin/printjob	root	-rwxr-xr-x	42667
PBS_EXEC/bin/qalter	root	-rwxr-xr-x	210723
PBS_EXEC/bin/qdel	root	-rwxr-xr-x	164949
PBS_EXEC/bin/qdisable	root	-rwxr-xr-x	139559
PBS_EXEC/bin/qenable	root	-rwxr-xr-x	139558
PBS_EXEC/bin/qhold	root	-rwxr-xr-x	165368
PBS_EXEC/bin/qmgr	root	-rwxr-xr-x	202526
PBS_EXEC/bin/qmove	root	-rwxr-xr-x	160932
PBS_EXEC/bin/qmsg	root	-rwxr-xr-x	160408
PBS_EXEC/bin/qorder	root	-rwxr-xr-x	146393
PBS_EXEC/bin/qrerun	root	-rwxr-xr-x	157228
PBS_EXEC/bin/qrls	root	-rwxr-xr-x	165361
PBS_EXEC/bin/qrun	root	-rwxr-xr-x	160978
PBS_EXEC/bin/qselect	root	-rwxr-xr-x	163266
PBS_EXEC/bin/qsig	root	-rwxr-xr-x	160083
PBS_EXEC/bin/qstart	root	-rwxr-xr-x	139589
PBS_EXEC/bin/qstat	root	-rwxr-xr-x	207532
PBS_EXEC/bin/qstop	root	-rwxr-xr-x	139584
PBS_EXEC/bin/qsub	root	-rwxr-xr-x	275460
PBS_EXEC/bin/qterm	root	-rwxr-xr-x	132188
PBS_EXEC/bin/tracejob	root	-rwxr-xr-x	64730
PBS_EXEC/bin/xpbs	root	-rwxr-xr-x	817

Appendix C: File Listing

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/bin/xpbsmon	root	-rwxr-xr-x	817
PBS_EXEC/etc	root	drwxr-xr-x	4096
PBS_EXEC/etc/au-nodeupdate.pl	root	-rw-r--r--	
PBS_EXEC/etc/pbs_dedicated	root	-rw-r--r--	557
PBS_EXEC/etc/pbs_holidays	root	-rw-r--r--	1173
PBS_EXEC/etc/pbs_init.d	root	-rwx------	5382
PBS_EXEC/etc/pbs_postinstall	root	-rwx------	10059
PBS_EXEC/etc/pbs_habitat	root	-rwx------	10059
PBS_EXEC/etc/pbs_resource_group	root	-rw-r--r--	657
PBS_EXEC/etc/pbs_sched_config	root	-r--r--r--	9791
PBS_EXEC/etc/pbs_setlicense	root	-rwx------	2118
PBS_EXEC/include	root	drwxr-xr-x	4096
PBS_EXEC/include/pbs_error.h	root	-r--r--r--	7543
PBS_EXEC/include/pbs_ifl.h	root	-r--r--r--	17424
PBS_EXEC/include/rm.h	root	-r--r--r--	740
PBS_EXEC/include/tm.h	root	-r--r--r--	2518
PBS_EXEC/include/tm_.h	root	-r--r--r--	2236
PBS_EXEC/lib	root	drwxr-xr-x	4096
PBS_EXEC/lib/libattr.a	root	-rw-r--r--	390274
PBS_EXEC/lib/libcmds.a	root	-rw-r--r--	328234
PBS_EXEC/lib/liblog.a	root	-rw-r--r--	101230
PBS_EXEC/lib/libnet.a	root	-rw-r--r--	145968
PBS_EXEC/lib/libpbs.a	root	-rw-r--r--	1815486

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/libsite.a	root	-rw-r--r--	132906
PBS_EXEC/lib/MPI	root	drwxr-xr-x	4096
PBS_EXEC/lib/MPI/ pbsrun.ch_gm.init.in	root	-rw-r--r--	9924
PBS_EXEC/lib/MPI/ pbsrun.ch_mx.init.in	root	-rw-r--r--	9731
PBS_EXEC/lib/MPI/ pbsrun.gm_mpd.init.in	root	-rw-r--r--	10767
PBS_EXEC/lib/MPI/ pbsrun.intelmpi.init.in	root	-rw-r--r--	10634
PBS_EXEC/lib/MPI/ pbsrun.mpich2.init.in	root	-rw-r--r--	10694
PBS_EXEC/lib/MPI/ pbsrun.mx_mpd.init.in	root	-rw-r--r--	10770
PBS_EXEC/lib/MPI/sgiMPI.awk	root	-rw-r--r--	6564
PBS_EXEC/lib/pbs_sched.a	root	-rw-r--r--	822026
PBS_EXEC/lib/pm	root	drwxr--r--	4096
PBS_EXEC/lib/pm/PBS.pm	root	-rw-r--r--	3908
PBS_EXEC/lib/xpbs	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbs/pbs_acctname.tk	root	-rw-r--r--	3484
PBS_EXEC/lib/xpbs/ pbs_after_depend.tk	root	-rw-r--r--	8637
PBS_EXEC/lib/xpbs/pbs_auto_upd.tk	root	-rw-r--r--	3384
PBS_EXEC/lib/xpbs/ pbs_before_depend.tk	root	-rw-r--r--	8034
PBS_EXEC/lib/xpbs/pbs_bin	root	drwxr-xr-x	4096

Appendix C: File Listing

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/pbs_bin/xpbs_datadump	root	-rwxr-xr-x	190477
PBS_EXEC/lib/xpbs/pbs_bin/xpbs_scriptload	root	-rwxr-xr-x	173176
PBS_EXEC/lib/xpbs/pbs_bindings.tk	root	-rw-r--r--	26029
PBS_EXEC/lib/xpbs/pbs_bitmaps	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbs/pbs_bitmaps/Downarrow.bmp	root	-rw-r--r--	299
PBS_EXEC/lib/xpbs/pbs_bitmaps/Uparrow.bmp	root	-rw-r--r--	293
PBS_EXEC/lib/xpbs/pbs_bitmaps/curve_down_arrow.bmp	root	-rw-r--r--	320
PBS_EXEC/lib/xpbs/pbs_bitmaps/curve_up_arrow.bmp	root	-rw-r--r--	314
PBS_EXEC/lib/xpbs/pbs_bitmaps/cyclist-only.xbm	root	-rw-r--r--	2485
PBS_EXEC/lib/xpbs/pbs_bitmaps/hourglass.bmp	root	-rw-r--r--	557
PBS_EXEC/lib/xpbs/pbs_bitmaps/iconize.bmp	root	-rw-r--r--	287
PBS_EXEC/lib/xpbs/pbs_bitmaps/logo.bmp	root	-rw-r--r--	67243
PBS_EXEC/lib/xpbs/pbs_bitmaps/maximize.bmp	root	-rw-r--r--	287
PBS_EXEC/lib/xpbs/pbs_bitmaps/sm_down_arrow.bmp	root	-rw-r--r--	311
PBS_EXEC/lib/xpbs/pbs_bitmaps/sm_up_arrow.bmp	root	-rw-r--r--	305

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/pbs_box.tk	root	-rw-r--r--	25912
PBS_EXEC/lib/xpbs/pbs_button.tk	root	-rw-r--r--	18795
PBS_EXEC/lib/xpbs/pbs_checkpoint.tk	root	-rw-r--r--	6892
PBS_EXEC/lib/xpbs/pbs_common.tk	root	-rw-r--r--	25940
PBS_EXEC/lib/xpbs/pbs_concur.tk	root	-rw-r--r--	8445
PBS_EXEC/lib/xpbs/pbs_datetime.tk	root	-rw-r--r--	4533
PBS_EXEC/lib/xpbs/pbs_email_list.tk	root	-rw-r--r--	3094
PBS_EXEC/lib/xpbs/pbs_entry.tk	root	-rw-r--r--	12389
PBS_EXEC/lib/xpbs/pbs_fileselect.tk	root	-rw-r--r--	7975
PBS_EXEC/lib/xpbs/pbs_help	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbs/pbs_help/after_depend.hlp	root	-rw-r--r--	1746
PBS_EXEC/lib/xpbs/pbs_help/auto_update.hlp	root	-rw-r--r--	776
PBS_EXEC/lib/xpbs/pbs_help/before_depend.hlp	root	-rw-r--r--	1413
PBS_EXEC/lib/xpbs/pbs_help/concur.hlp	root	-rw-r--r--	1383
PBS_EXEC/lib/xpbs/pbs_help/datetime.hlp	root	-rw-r--r--	698
PBS_EXEC/lib/xpbs/pbs_help/delete.hlp	root	-rw-r--r--	632
PBS_EXEC/lib/xpbs/pbs_help/email.hlp	root	-rw-r--r--	986

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/pbs_help/fileselect.hlp	root	-rw-r--r--	1655
PBS_EXEC/lib/xpbs/pbs_help/hold.hlp	root	-rw-r--r--	538
PBS_EXEC/lib/xpbs/pbs_help/main.hlp	root	-rw-r--r--	15220
PBS_EXEC/lib/xpbs/pbs_help/message.hlp	root	-rw-r--r--	677
PBS_EXEC/lib/xpbs/pbs_help/misc.hlp	root	-rw-r--r--	4194
PBS_EXEC/lib/xpbs/pbs_help/modify.hlp	root	-rw-r--r--	6034
PBS_EXEC/lib/xpbs/pbs_help/move.hlp	root	-rw-r--r--	705
PBS_EXEC/lib/xpbs/pbs_help/notes.hlp	root	-rw-r--r--	3724
PBS_EXEC/lib/xpbs/pbs_help/preferences.hlp	root	-rw-r--r--	1645
PBS_EXEC/lib/xpbs/pbs_help/release.hlp	root	-rw-r--r--	573
PBS_EXEC/lib/xpbs/pbs_help/select.acctname.hlp	root	-rw-r--r--	609
PBS_EXEC/lib/xpbs/pbs_help/select.checkpoint.hlp	root	-rw-r--r--	1133
PBS_EXEC/lib/xpbs/pbs_help/select.hold.hlp	root	-rw-r--r--	544
PBS_EXEC/lib/xpbs/pbs_help/select.jobname.hlp	root	-rw-r--r--	600

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/pbs_help/select.owners.hlp	root	-rw-r--r--	1197
PBS_EXEC/lib/xpbs/pbs_help/select.priority.hlp	root	-rw-r--r--	748
PBS_EXEC/lib/xpbs/pbs_help/select.qtime.hlp	root	-rw-r--r--	966
PBS_EXEC/lib/xpbs/pbs_help/select.rerun.hlp	root	-rw-r--r--	541
PBS_EXEC/lib/xpbs/pbs_help/select.resources.hlp	root	-rw-r--r--	1490
PBS_EXEC/lib/xpbs/pbs_help/select.states.hlp	root	-rw-r--r--	562
PBS_EXEC/lib/xpbs/pbs_help/signal.hlp	root	-rw-r--r--	675
PBS_EXEC/lib/xpbs/pbs_help/staging.hlp	root	-rw-r--r--	3702
PBS_EXEC/lib/xpbs/pbs_help/submit.hlp	root	-rw-r--r--	9721
PBS_EXEC/lib/xpbs/pbs_help/terminate.hlp	root	-rw-r--r--	635
PBS_EXEC/lib/xpbs/pbs_help/trackjob.hlp	root	-rw-r--r--	2978
PBS_EXEC/lib/xpbs/pbs_hold.tk	root	-rw-r--r--	3539
PBS_EXEC/lib/xpbs/pbs_jobname.tk	root	-rw-r--r--	3375
PBS_EXEC/lib/xpbs/pbs_listbox.tk	root	-rw-r--r--	10544
PBS_EXEC/lib/xpbs/pbs_main.tk	root	-rw-r--r--	24147
PBS_EXEC/lib/xpbs/pbs_misc.tk	root	-rw-r--r--	14526

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/pbs_owners.tk	root	-rw-r--r--	4509
PBS_EXEC/lib/xpbs/pbs_pbs.tcl	root	-rw-r--r--	52524
PBS_EXEC/lib/xpbs/pbs_pref.tk	root	-rw-r--r--	3445
PBS_EXEC/lib/xpbs/ pbs_preferences.tcl	root	-rw-r--r--	4323
PBS_EXEC/lib/xpbs/pbs_prefsave.tk	root	-rw-r--r--	1378
PBS_EXEC/lib/xpbs/pbs_priority.tk	root	-rw-r--r--	4434
PBS_EXEC/lib/xpbs/pbs_qalter.tk	root	-rw-r--r--	35003
PBS_EXEC/lib/xpbs/pbs_qdel.tk	root	-rw-r--r--	3175
PBS_EXEC/lib/xpbs/pbs_qhold.tk	root	-rw-r--r--	3676
PBS_EXEC/lib/xpbs/pbs_qmove.tk	root	-rw-r--r--	3326
PBS_EXEC/lib/xpbs/pbs_qmsg.tk	root	-rw-r--r--	4032
PBS_EXEC/lib/xpbs/pbs_qrls.tk	root	-rw-r--r--	3674
PBS_EXEC/lib/xpbs/pbs_qsig.tk	root	-rw-r--r--	5171
PBS_EXEC/lib/xpbs/pbs_qsub.tk	root	-rw-r--r--	37466
PBS_EXEC/lib/xpbs/pbs_qterm.tk	root	-rw-r--r--	3204
PBS_EXEC/lib/xpbs/pbs_qtime.tk	root	-rw-r--r--	5790
PBS_EXEC/lib/xpbs/pbs_rerun.tk	root	-rw-r--r--	2802
PBS_EXEC/lib/xpbs/pbs_res.tk	root	-rw-r--r--	4807
PBS_EXEC/lib/xpbs/pbs_spinbox.tk	root	-rw-r--r--	7144
PBS_EXEC/lib/xpbs/pbs_staging.tk	root	-rw-r--r--	12183
PBS_EXEC/lib/xpbs/pbs_state.tk	root	-rw-r--r--	3657
PBS_EXEC/lib/xpbs/pbs_text.tk	root	-rw-r--r--	2738

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/pbs_trackjob.tk	root	-rw-r--r--	13605
PBS_EXEC/lib/xpbs/pbs_wmgr.tk	root	-rw-r--r--	1428
PBS_EXEC/lib/xpbs/tclIndex	root	-rw-r--r--	19621
PBS_EXEC/lib/xpbs/xpbs.src.tk	root	-rwxr-xr-x	9666
PBS_EXEC/lib/xpbs/xpbsrc	root	-rw-r--r--	2986
PBS_EXEC/lib/xpbsmon	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbsmon/ pbs_auto_upd.tk	root	-rw-r--r--	3281
PBS_EXEC/lib/xpbsmon/ pbs_bindings.tk	root	-rw-r--r--	9288
PBS_EXEC/lib/xpbsmon/pbs_bitmaps	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbsmon/ pbs_bitmaps/cyclist-only.xbm	root	-rw-r--r--	2485
PBS_EXEC/lib/xpbsmon/ pbs_bitmaps/hourglass.bmp	root	-rw-r--r--	557
PBS_EXEC/lib/xpbsmon/ pbs_bitmaps/iconize.bmp	root	-rw-r--r--	287
PBS_EXEC/lib/xpbsmon/ pbs_bitmaps/logo.bmp	root	-rw-r--r--	67243
PBS_EXEC/lib/xpbsmon/ pbs_bitmaps/maximize.bmp	root	-rw-r--r--	287
PBS_EXEC/lib/xpbsmon/pbs_box.tk	root	-rw-r--r--	15607
PBS_EXEC/lib/xpbsmon/ pbs_button.tk	root	-rw-r--r--	7543
PBS_EXEC/lib/xpbsmon/ pbs_cluster.tk	root	-rw-r--r--	44406

Appendix C: File Listing

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbsmon/pbs_color.tk	root	-rw-r--r--	5634
PBS_EXEC/lib/xpbsmon/ pbs_common.tk	root	-rw-r--r--	5716
PBS_EXEC/lib/xpbsmon/ pbs_dialog.tk	root	-rw-r--r--	8398
PBS_EXEC/lib/xpbsmon/pbs_entry.tk	root	-rw-r--r--	10697
PBS_EXEC/lib/xpbsmon/pbs_expr.tk	root	-rw-r--r--	6163
PBS_EXEC/lib/xpbsmon/pbs_help	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbsmon/pbs_help/ auto_update.hlp	root	-rw-r--r--	624
PBS_EXEC/lib/xpbsmon/pbs_help/ main.hlp	root	-rw-r--r--	15718
PBS_EXEC/lib/xpbsmon/pbs_help/ notes.hlp	root	-rw-r--r--	296
PBS_EXEC/lib/xpbsmon/pbs_help/ pref.hlp	root	-rw-r--r--	1712
PBS_EXEC/lib/xpbsmon/pbs_help/ prefQuery.hlp	root	-rw-r--r--	4621
PBS_EXEC/lib/xpbsmon/pbs_help/ prefServer.hlp	root	-rw-r--r--	1409
PBS_EXEC/lib/xpbsmon/ pbs_listbox.tk	root	-rw-r--r--	10640
PBS_EXEC/lib/xpbsmon/pbs_main.tk	root	-rw-r--r--	6760
PBS_EXEC/lib/xpbsmon/pbs_node.tk	root	-rw-r--r--	60640
PBS_EXEC/lib/xpbsmon/pbs_pbs.tk	root	-rw-r--r--	7090
PBS_EXEC/lib/xpbsmon/pbs_pref.tk	root	-rw-r--r--	22117

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbsmon/pbs_preferences.tcl	root	-rw-r--r--	10212
PBS_EXEC/lib/xpbsmon/pbs_prefsave.tk	root	-rw-r--r--	1482
PBS_EXEC/lib/xpbsmon/pbs_spinbox.tk	root	-rw-r--r--	7162
PBS_EXEC/lib/xpbsmon/pbs_system.tk	root	-rw-r--r--	47760
PBS_EXEC/lib/xpbsmon/pbs_wmgr.tk	root	-rw-r--r--	1140
PBS_EXEC/lib/xpbsmon/tclIndex	root	-rw-r--r--	30510
PBS_EXEC/lib/xpbsmon/xpbsmon.src.tk	root	-rwxr-xr-x	13999
PBS_EXEC/lib/xpbsmon/xpbsmonrc	root	-rw-r--r--	3166
PBS_EXEC/man	root	drwxr-xr-x	4096
PBS_EXEC/man/man1	root	drwxr-xr-x	4096
PBS_EXEC/man/man1/nqs2pbs.1B	root	-rw-r--r--	3276
PBS_EXEC/man/man1/pbs.1B	root	-rw-r--r--	5376
PBS_EXEC/man/man1/pbs_rdel.1B	root	-rw-r--r--	2342
PBS_EXEC/man/man1/pbs_rstat.1B	root	-rw-r--r--	2682
PBS_EXEC/man/man1/pbs_rsub.1B	root	-rw-r--r--	9143
PBS_EXEC/man/man1/pbsdsh.1B	root	-rw-r--r--	2978
PBS_EXEC/man/man1/qalter.1B	root	-rw-r--r--	21569
PBS_EXEC/man/man1/qdel.1B	root	-rw-r--r--	3363
PBS_EXEC/man/man1/qhold.1B	root	-rw-r--r--	4323

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/man/man1/qmove.1B	root	-rw-r--r--	3343
PBS_EXEC/man/man1/qmsg.1B	root	-rw-r--r--	3244
PBS_EXEC/man/man1/qorder.1B	root	-rw-r--r--	3028
PBS_EXEC/man/man1/qrerun.1B	root	-rw-r--r--	2965
PBS_EXEC/man/man1/qrls.1B	root	-rw-r--r--	3927
PBS_EXEC/man/man1/qselect.1B	root	-rw-r--r--	12690
PBS_EXEC/man/man1/qsig.1B	root	-rw-r--r--	3817
PBS_EXEC/man/man1/qstat.1B	root	-rw-r--r--	15274
PBS_EXEC/man/man1/qsub.1B	root	-rw-r--r--	36435
PBS_EXEC/man/man1/xpbs.1B	root	-rw-r--r--	26956
PBS_EXEC/man/man1/xpbsmon.1B	root	-rw-r--r--	26365
PBS_EXEC/man/man3	root	drwxr-xr-x	4096
PBS_EXEC/man/man3/ pbs_alterjob.3B	root	-rw-r--r--	5475
PBS_EXEC/man/man3/ pbs_connect.3B	root	-rw-r--r--	3493
PBS_EXEC/man/man3/ pbs_default.3B	root	-rw-r--r--	2150
PBS_EXEC/man/man3/pbs_deljob.3B	root	-rw-r--r--	3081
PBS_EXEC/man/man3/ pbs_disconnect.3B	root	-rw-r--r--	1985
PBS_EXEC/man/man3/ pbs_geterrmsg.3B	root	-rw-r--r--	2473
PBS_EXEC/man/man3/ pbs_holdjob.3B	root	-rw-r--r--	3006

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/man/man3/pbs_manager.3B	root	-rw-r--r--	4337
PBS_EXEC/man/man3/pbs_movejob.3B	root	-rw-r--r--	3220
PBS_EXEC/man/man3/pbs_msgjob.3B	root	-rw-r--r--	2912
PBS_EXEC/man/man3/pbs_orderjob.3B	root	-rw-r--r--	2526
PBS_EXEC/man/man3/pbs_rerunjob.3B	root	-rw-r--r--	2531
PBS_EXEC/man/man3/pbs_rescreserve.3B	root	-rw-r--r--	4125
PBS_EXEC/man/man3/pbs_rlsjob.3B	root	-rw-r--r--	3043
PBS_EXEC/man/man3/pbs_runjob.3B	root	-rw-r--r--	3484
PBS_EXEC/man/man3/pbs_selectjob.3B	root	-rw-r--r--	7717
PBS_EXEC/man/man3/pbs_sigjob.3B	root	-rw-r--r--	3108
PBS_EXEC/man/man3/pbs_stagein.3B	root	-rw-r--r--	3198
PBS_EXEC/man/man3/pbs_statjob.3B	root	-rw-r--r--	4618
PBS_EXEC/man/man3/pbs_statnode.3B	root	-rw-r--r--	3925
PBS_EXEC/man/man3/pbs_statque.3B	root	-rw-r--r--	4009
PBS_EXEC/man/man3/pbs_statserver.3B	root	-rw-r--r--	3674

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/man/man3/pbs_submit.3B	root	-rw-r--r--	6320
PBS_EXEC/man/man3/pbs_submitresv.3B	root	-rw-r--r--	3878
PBS_EXEC/man/man3/pbs_terminate.3B	root	-rw-r--r--	3322
PBS_EXEC/man/man3/rpp.3B	root	-rw-r--r--	6476
PBS_EXEC/man/man3/tm.3B	root	-rw-r--r--	11062
PBS_EXEC/man/man7	root	drwxr-xr-x	4096
PBS_EXEC/man/man7/pbs_job_attributes.7B	root	-rw-r--r--	15920
PBS_EXEC/man/man7/pbs_node_attributes.7B	root	-rw-r--r--	7973
PBS_EXEC/man/man7/pbs_queue_attributes.7B	root	-rw-r--r--	11062
PBS_EXEC/man/man7/pbs_resources.7B	root	-rw-r--r--	22124
PBS_EXEC/man/man7/pbs_resv_attributes.7B	root	-rw-r--r--	11662
PBS_EXEC/man/man7/pbs_server_attributes.7B	root	-rw-r--r--	14327
PBS_EXEC/man/man8	root	drwxr-xr-x	4096
PBS_EXEC/man/man8/mpiexec.8B	root	-rw-r--r--	4701
PBS_EXEC/man/man8/pbs-report.8B	root	-rw-r--r--	19221
PBS_EXEC/man/man8/pbs_attach.8B	root	-rw-r--r--	3790
PBS_EXEC/man/man8/pbs_hostn.8B	root	-rw-r--r--	2781

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/man/man8/pbs_idled.8B	root	-rw-r--r--	2628
PBS_EXEC/man/man8/pbs_lamboot.8B	root	-rw-r--r--	2739
PBS_EXEC/man/man8/pbs_migrate_users.8B	root	-rw-r--r--	2519
PBS_EXEC/man/man8/pbs_mom.8B	root	-rw-r--r--	23496
PBS_EXEC/man/man8/pbs_mom_globus.8B	root	-rw-r--r--	11054
PBS_EXEC/man/man8/pbs_mpihp.8B	root	-rw-r--r--	4120
PBS_EXEC/man/man8/pbs_mpilam.8B	root	-rw-r--r--	2647
PBS_EXEC/man/man8/pbs_mpirun.8B	root	-rw-r--r--	3130
PBS_EXEC/man/man8/pbs_password.8B	root	-rw-r--r--	3382
PBS_EXEC/man/man8/pbs_poe.8B	root	-rw-r--r--	3973
PBS_EXEC/man/man8/pbs_probe.8B	root	-rw-r--r--	3344
PBS_EXEC/man/man8/pbs_sched_cc.8B	root	-rw-r--r--	6731
PBS_EXEC/man/man8/pbs_server.8B	root	-rw-r--r--	7914
PBS_EXEC/man/man8/pbs_tclsh.8B	root	-rw-r--r--	2475
PBS_EXEC/man/man8/pbs_tmrsh.8B	root	-rw-r--r--	3556
PBS_EXEC/man/man8/pbs_wish.8B	root	-rw-r--r--	2123
PBS_EXEC/man/man8/pbsfs.8B	root	-rw-r--r--	3703
PBS_EXEC/man/man8/pbsnodes.8B	root	-rw-r--r--	3441

Appendix C: File Listing

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/man/man8/pbsrun.8B	root	-rw-r--r--	20937
PBS_EXEC/man/man8/pbsrun_unwrap.8B	root	-rw-r--r--	2554
PBS_EXEC/man/man8/pbsrun_wrap.8B	root	-rw-r--r--	3855
PBS_EXEC/man/man8/printjob.8B	root	-rw-r--r--	2823
PBS_EXEC/man/man8/qdisable.8B	root	-rw-r--r--	3104
PBS_EXEC/man/man8/qenable.8B	root	-rw-r--r--	2937
PBS_EXEC/man/man8/qmgr.8B	root	-rw-r--r--	7282
PBS_EXEC/man/man8/qrun.8B	root	-rw-r--r--	2850
PBS_EXEC/man/man8/qstart.8B	root	-rw-r--r--	2966
PBS_EXEC/man/man8/qstop.8B	root	-rw-r--r--	2963
PBS_EXEC/man/man8/qterm.8B	root	-rw-r--r--	4839
PBS_EXEC/man/man8/tracejob.8B	root	-rw-r--r--	4664
PBS_EXEC/sbin	root	drwxr-xr-x	4096
PBS_EXEC/sbin/pbs-report	root	-rwxr-xr-x	68296
PBS_EXEC/sbin/pbs_demux	root	-rwxr-xr-x	38688
PBS_EXEC/sbin/pbs_idled	root	-rwxr-xr-x	99373
PBS_EXEC/sbin/pbs_iff	root	-rwsr-xr-x	133142
PBS_EXEC/sbin/pbs_mom	root	-rwx------	839326
PBS_EXEC/sbin/pbs_mom.cpuset	root	-rwx------	0
PBS_EXEC/sbin/pbs_mom.standard	root	-rwx------	0
PBS_EXEC/sbin/pbs_probe	root	-rwsr-xr-x	83108
PBS_EXEC/sbin/pbs_rcp	root	-rwsr-xr-x	75274

Table 1:

Directory / File	Owner	Permission	Average Size
PBS_EXEC/sbin/pbs_sched	root	-rwx------	705478
PBS_EXEC/sbin/pbs_server	root	-rwx------	1133650
PBS_EXEC/sbin/pbsfs	root	-rwxr-xr-x	663707
PBS_EXEC/tcltk	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/bin	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/bin/tclsh8.3	root	-rw-r--r--	552763
PBS_EXEC/tcltk/bin/wish8.3	root	-rw-r--r--	1262257
PBS_EXEC/tcltk/include	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/include/tcl.h	root	-rw-r--r--	57222
PBS_EXEC/tcltk/include/tclDecls.h	root	-rw-r--r--	123947
PBS_EXEC/tcltk/include/tk.h	root	-rw-r--r--	47420
PBS_EXEC/tcltk/include/tkDecls.h	root	-rw-r--r--	80181
PBS_EXEC/tcltk/lib	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/lib/libtcl8.3.a	root	-rw-r--r--	777558
PBS_EXEC/tcltk/lib/libtclstub8.3.a	root	-rw-r--r--	1832
PBS_EXEC/tcltk/lib/libtk8.3.a	root	-rw-r--r--	1021024
PBS_EXEC/tcltk/lib/libtkstub8.3.a	root	-rw-r--r--	3302
PBS_EXEC/tcltk/lib/tcl8.3	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/lib/tclConfig.sh	root	-rw-r--r--	7076
PBS_EXEC/tcltk/lib/tk8.3	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/lib/tkConfig.sh	root	-rw-r--r--	3822
PBS_EXEC/tcltk/license.terms	root	-rw-r--r--	2233

Appendix D: Log Messages

The server, scheduler and MOM all write messages to their log files. Which messages are written depends upon each daemon's event mask. See section 6.17.1 "PBS Events" on page 354, section 6.17.2 "Event Logfiles" on page 356. and section 6.17.3 "Event Logfile Format" on page 357.

A few log messages are listed here.

RPP Retries

Logs	Server, scheduler, MOM
Level	0002; DEBUG

Form	date; time;event type; reporting daemon; event class; rpp_stats; total (pkts=<packets>, retries=<retries>, fails=<fails>) last <number of seconds> secs (pkts=<pack-ets>,retries=<retries>, fails=<fails>)
Example	03/22/2006 15:20:44;0002; pbs_mom; Svr;rpp_stats; total (pkts=4321, retries=25, fails=3) last 3621 secs (pkts=43, retries=2, fails=0)
Explanation	RPP packet retries, reported both for total number since daemon start (“total”) and since last log message (“last <seconds> secs”). Logged at most once per hour unless this hour’s retry count is 0. The number of seconds since the previous log message is shown in “last <seconds> secs”. pkts: number of RPP packets sent. In “total” group, this is since daemon start (in example, 4321). In “last” group, this is since previous log message (in example, 43). retries: number of RPP data packet retries. In “total” group, this is since daemon start (in example, 25). In “last” group, this is since previous log message (in example, 2). fails: number of failures reported to the caller of the RPP function. In “total” group, this is since daemon start (in example, 3). In “last” group, this is since previous log message (in example, 0). No log message if the number of fails and retries are zero.

cput and mem Logged by Mother Superior

Logs	Mother Superior

Level	0100
Form	Date; Time; event class; reporting daemon; Job; Job ID; Hostname; cput; mem
Example	07/02/2007 19:47:14;0100;pbs_mom;Job;40.pepsi;pepsi cput= 0:00:00 mem=4756kb
Explanation	On job exit, Mother superior logs the amount of cput and mem used by this job on each node.

MOM Adds $clienthost Address

Logs	MOM
Level	Event level 0x2, PBSE_SYSTEM, event class Server
Form	Adding IP address XXX.XXX.XXX.XXX as authorized
Example	Adding IP address 127.0.0.1 as authorized
Explanation	When MOM starts up, she logs the addresses associated with a host listed in Mom's config file in $clienthost statements. When MOM receives the list from the Server, addresses associated with other MOMs in the PBS complex will be listed. This occurs as soon as MOM and the Server establish communication and again whenever a node goes down and comes back up, or there is a change to the list of execution hosts (node added to or deleted from the complex). That event and the associated logging may occur at any time.

Scheduler: Job is Invalid

Logs	Scheduler
Level	DEBUG, which is in the default set of levels
Form	Job is invalid - ignoring for this cycle

Example	Job is invalid - ignoring for this cycle
Explanation	Job failed a validity check such as 1) no egroup, euser, select, place, 2) in peer scheduling, pulling server is not a manager for furnishing server, 3) internal scheduler memory failure

Scheduler: Can't find subjob in simulated universe

Logs	Scheduler
Level	DEBUG
Form	can't find new subjob in simulated universe
Example	can't find new subjob in simulated universe
Explanation	This means that when backfilling around a job array, we can run into an error case. The error case we're handling here is that we have simulated the future in a simulated universe. In the simulated universe, we've spawned and run a subjob. Now we're trying to find it so we can do the same thing in the real universe. The simulated subjob can't be found.

Scheduler: Message Indicating Whether It Is Prime Time

Logs	Scheduler
Level	DEBUG2 (256)
Form	"It is *P*. It will end in XX seconds at MM/DD/YYYY HH:MM:SS"
Example	"It is prime time. It will end in 29 seconds at 03/10/2007 09:29:31"
Explanation	The scheduler is declaring whether the current time is prime time or non-prime time. The scheduler is stating when this period of prime time or non-prime time will end.

Jobs that Can Never Run

Logs	Scheduler
Level	DEBUG
Form	“resource request is impossible to solve: job will never run”
Example	“resource request is impossible to solve: job will never run”
Explanation	The “most deserving” job can never run. Only printed when backfilling is on.

Resource Permission Flag Error

Table 1:

Logs	Server
Level	511; DEBUG1 & DEBUG2
Form	“It is invalid to set both flags 'r' and 'i'. Flag 'r' will be ignored.”
Example	“It is invalid to set both flags 'r' and 'i'. Flag 'r' will be ignored.”
Explanation	The “i” and “r” flags are incompatible. The “i” flag takes precedence.

Error During Evaluation of Tunable Formula

Table 2:

Logs	Scheduler
Level	DEBUG2
Form	1234.mars;Formula evaluation for job had an error. Zero value will be used

Table 2:

Example	1234.mars;Formula evaluation for job had an error. Zero value will be used
Explanation	Tunable formula produced error when evaluated.

Creation of Job-specific Directory

Table 3:

Logs	MOM
Level	PBSEVENT_JOB
Form	“created the job directory <jobdir_root><unique_job_dir_name>”
Example	“created the job directory /Users/user1/pbsjobs/345.myhost”
Explanation	PBS created a job-specific execution and staging directory

Failure to Create Job-specific Directory

Table 4:

Logs	MOM
Level	PBSEVENT_ERROR
Form	“unable to create the job directory <unique_job_dir_name>”
Example	“unable to create the job directory /Users/user1/pbsjobs/345.myhost”
Explanation	The MOM was unable to create the job-specific staging and execution directory

Failure toValidate MOM’s $jobdir_root

Table 5:

Logs	MOM
Level	PBSEVENT_ERROR
Form	"<file>[<linenum>]command “$jobdir_root <full path>” failed, aborting"
Example	"config[3] command “$jobdir_root /foodir” failed, aborting"
Explanation	$jobdir_root exists in the MOM’s configuration file, and the MOM was unable to validate $jobdir_root; MOM has aborted

Job Eligible Time

Table 6:

Logs	Server
Level	DEBUG3
Form	MM/DD/YYYY hh:mm:ss;log event;Server@hostname;Job;jobID;job accrued 23 secs of <previous sample type>, new accrue_type=<next_sample_type>, eligible_time=<current amount of eligible_time>
Example	08/07/2007 13:xx:yy;0040;Server@myhost;Job;163.myhost;job accrued 23 secs of eligible_time, new accrue_type=run_time, eligible_time=00:00:23
Explanation	Previous sample was of eligible time; next sample will be of run_time, job has accrued 23 seconds of eligible_time.

Invalid Syntax for Standing Reservation

Logs	Server

Level	PBSEVENT_DEBUG
Form	pbs_rsub error: Undefined iCalendar syntax
Example	pbs_rsub error: Undefined iCalendar syntax
Explanation	Invalid syntax given to pbs_rsub for recurrence rule for standing reservation

Appendix D: Log Messages

Appendix E: License Agreement

Altair Engineering, Inc.
Software License Agreement

This License Agreement is a legal agreement between Altair Engineering, Inc. ("Altair") and you ("Licensee") governing the terms of use of the Altair Software. Before you may download or use the Software, your consent to the following terms and conditions is required by clicking on the 'I Accept" button. If you do not have the authority to bind your organization to these terms and conditions, you must click on the button that states "I do not accept" and then have an authorized party in your organization consent to these terms. In the event that your organization and Altair have a master software license agreement, mutually agreed upon in writing, in place at the time of your execution of this agreement, the terms of the master agreement shall govern.

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:
Documentation. Documentation provided by Altair on any media for use with the Software.
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.
License Log File. A computer file providing usage information on the Software as gathered by the Software.
License Management System. The license management system that accompanies the Software and limits its use in accordance with the usage permitted under 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 on the Order Form.
License Units. A parameter used by the License Management System to determine the 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.
Maintenance Release. Any release of the Software 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.
Proprietary Rights Notices. Patent, copyright, trademark or other proprietary rights notices applied to the Software, Documentation or the packaging or media of same.
Software. The software identified in the Order Form and any Updates or Maintenance Releases.

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 initial term of this Agreement or any renewal term. Annual licenses shall have a 12-month term of use. Paid-up, or perpetual licenses, shall have a term of twenty-five years.
Update. A new version of the Software made generally available by Altair to its Licensee that includes additional features or functionalities but is substantially the same computer code as the existing Software.

2. PAYMENT. Licensee shall pay in full the fee for licensed Software and Support within thirty (30) days of receipt of the invoice. Past due fees shall bear interest at the maximum legal rate. Altair may condition its delivery of any Maintenance Release or Update to Licensee on Licensee's having paid all amounts then owed to Altair. Fees do not include taxes or duties and Licensee is responsible for paying (or for reimbursing Altair if Altair is required to pay) any federal, state or local taxes, or duties imposed on this License or the possession or use by Licensee of the Software excluding, however, all taxes on or measured by Altair's net income. Altair shall be entitled to its reasonable costs of collection (including attorneys fees and interest) if license fees are not paid to it on a timely basis.

3. TERM. Unless terminated earlier in accordance with the provisions of this Agreement, this Agreement will be in force for a period as stated on the Order Form. For annual licenses or Support provided for perpetual licenses, renewal shall be automatic for a 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 procured by Licensee may be made coterminous at the request of Licensee and the consent of Altair.

4. 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 Software 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 Software on Licensed Workstations in accordance with the License Management System 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 identified on the Order Form and for the term identified on the Order Form; c) make backup copies of the Software, provided that Altair's Proprietary Rights Notices are reproduced on each such backup copy; d) freely modify and use Templates, 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. 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 to provide itself with 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).

5. **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 Software or Documentation, except as expressly authorized in this Agreement; b) use any back-up copies of the Software 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 for any reason; d) sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the Software or Documentation 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 Software, such as through a service bureau, wide area network, Internet location or time-sharing arrangement except as expressly provided in Section 4(b); g) remove any Proprietary Rights Notices from the Software; h) disable or circumvent the License Management System provided with the Software; or (i) develop, test or support software of Licensee or third parties.

6. **OWNERSHIP AND CONFIDENTIALITY**. Licensee acknowledges that all applicable rights in patents, copyrights, trademarks, service marks, and trade secrets embodied in the Software and Documentation are owned by Altair and/or its Suppliers. Licensee further acknowledges that the Software and Documentation, and all copies thereof, are and shall remain the sole and exclusive property of Altair and/or its Suppliers. This Agreement is a license and not a sale of the Software. Altair retains all rights in the Software and Documentation not expressly granted to Licensee herein. Licensee acknowledges that the Software and accompanying Documentation are confidential and constitute valuable assets and trade secrets of Altair and/or its Suppliers. Licensee agrees to take the precautions necessary to protect and maintain the confidentiality of the Software and Documentation, 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 Software. If Licensee is required by any governmental authority or court of law to disclose Altair's 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 5 and Section 6 of this Agreement shall survive termination of this Agreement for any reason. Altair's Suppliers, 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 shall keep confidential all Licensee information provided to Altair in order that Altair may provide Support to Licensee shall be kept confidential and used only for the purpose of assisting Licensee in its use of the licensed Software.

7. **MAINTENANCE AND SUPPORT**. Maintenance. Altair will provide Licensee at no additional charge for annual licenses, and for a fee for paid-up licenses, with any Maintenance Releases and Updates of the Software or Documentation that are generally released by Altair during the term of this Agreement, except that this shall not apply to any 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 must install Maintenance Releases and Updates promptly after receipt from Altair. Maintenance Releases and Updates are Software subject to this Agreement. Altair shall only be obligated to provide support and maintenance for the most current release of the Software and its 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 will 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 Monday through Friday (excluding holidays) from 8:00 a.m. to 5:00 p.m local time in the Global Zone where licensed. 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; (b) any version of the Software other than the current version of the Software or the immediately previous version; (c) Software problems causes 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.

8. 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 and this Agreement. Supplier software included in the Software and provided to Licensee shall be warranted as stated by the Supplier. Copies of the Suppliers' terms and conditions for software 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 SOFTWARE WILL MEET LICENSEE'S REQUIREMENTS OR THAT ITS OPERATION WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT IT WILL BE COMPATIBLE WITH ANY PARTICULAR HARDWARE OR SOFTWARE. ALTAIR EXCLUDES AND DIS**CLAIMS 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 SOFTWARE 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 SOFTWARE.

9. INDEMNITY. Altair will defend, 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 based on Supplier 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 license binding 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 reasonably 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 9, 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 Licenses 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 9 states Altair's entire liability to Licensee in the event a Claim is made.

10. 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 8 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 8 for ninety (90) days after delivery to Licensee. The warranties hereunder are void if the Software has been misused or improperly installed, 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), except as provided in Section 9, 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. ALTAIR AND ITS SUPPLIERS 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 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 from any and all claims, including reasonable attorney's fees, resulting from, or in connection with Licensee's use of the Software. No 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.

11. 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.

12. UNITED STATES GOVERNMENT RESTRICTED RIGHTS. This section applies to all acquisitions of the Software by or for the United States government. By accepting delivery of the Software, the government hereby agrees that the Software qualifies 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 Software 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 Software for a full refund.

For procurements governed by DFARS Part 227.72 (OCT 1998), HyperWorks Software 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.

13. 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.

14. <u>GENERAL PROVISIONS</u>. Export Controls. Licensee acknowledges that the Software may be subject to the export control laws and regulations of the United States and any amendments thereof. Licensee agrees that Licensee will not directly or indirectly export the Software into any country or use the Software in any manner except in compliance with all applicable U.S. 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 notices shall be deemed given when actually received. **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 Purchase Order or comparable document, the terms of this Agreement shall prevail. Moreover, each party agrees any additional terms on any Purchase Order 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. **Entire Agreement**. This Agreement and the Order Form(s) 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.

Index

Index

Index

Index

Index

Index

Index

Index