

Table of Contents

Running Jobs with PBS	1
<u>Portable Batch System (PBS): Overview</u>	1
<u>Job Accounting</u>	3
<u>Job Accounting Utilities</u>	4
<u>Multiple GIDs and Charging to a specific GID</u>	6
<u>Commonly Used PBS Commands</u>	7
<u>Commonly Used QSUB Options in PBS Scripts or in the QSUB Command Line</u>	9
<u>New Features in PBS</u>	11
<u>Checkpointing and Restart</u>	13
<u>PBS Environment Variables</u>	14
<u>PBS Scheduling Policy</u>	15
<u>PBS exit codes</u>	18
Front-End Usage Guidelines	19
<u>Pleiades Front-End Usage Guidelines</u>	19
<u>Columbia Front-End Usage Guidelines</u>	21
PBS on Pleiades	22
<u>Overview</u>	22
<u>Queue Structure</u>	23
<u>Mission Shares Policy on Pleiades</u>	25
<u>Resources Request Examples</u>	27
<u>Default Variables Set by PBS</u>	30
<u>Sample PBS Script for Pleiades</u>	31
<u>Pleiades devel Queue</u>	32
PBS on Columbia	33
<u>Overview</u>	33
<u>Resources Request Examples</u>	34
<u>Default Variables Set by PBS</u>	36
<u>Sample PBS Script for Columbia</u>	37
Troubleshooting PBS Jobs	38
<u>Common Reasons for Being Unable to Submit Jobs</u>	38
<u>Common Reasons Why Jobs Won't Start</u>	40
<u>Using pdsh_gdb for Debugging Pleiades PBS Jobs</u>	44
Effective Use of PBS	45
<u>Streamlining PBS Job File Transfers from Pleiades to Lou</u>	45
<u>Avoiding Job Failure from Overfilling /PBS/spool</u>	46
<u>Running Multiple Serial Jobs to Reduce Wall-Time</u>	48
<u>Checking the Time Remaining in a PBS Job from a Fortran Code</u>	51
<u>Using GNU Parallel to Package Multiple Jobs in a Single PBS Job</u>	52

Running Jobs with PBS

Portable Batch System (PBS): Overview

All NAS facility supercomputers use the Portable Batch System (PBS) from Altair for batch job submission, job monitoring, and job management. Note that different systems may use different versions of PBS, so the available features may vary slightly from system to system.

Batch Jobs

Batch jobs run on compute nodes, not the front-end nodes. A PBS scheduler allocates blocks of compute nodes to jobs to provide exclusive access. You will submit batch jobs to run on one or more compute nodes using the `qsub` command from an interactive session on one of the front-end nodes (such as, `pfe[20-27]`, `bridge[1-4]` for Pleiades or `cfe2` for Columbia).

Normal batch jobs are typically run by submitting a script. A "jobid" is assigned after submission. When the resources you request become available, your job will execute on the compute nodes. When the job is complete, the PBS standard output and standard error of the job will be returned in files available to you.

Take careful note when porting job submission scripts from systems outside of the NAS environment or between the Pleiades and Columbia supercomputers you may need to make changes to your existing scripts to make them work properly on these systems.

Interactive Batch Mode

PBS also supports an interactive batch mode, using the `qsub -I`, where the input and output is connected to the user's terminal, but the scheduling of the job is still under control of the batch system.

Queues

The available queues on different systems vary, but all typically have constraints on maximum wall-time and/or the number of CPUs allowed for a job. Some queues may also have other constraints or be restricted to serving certain users or groups. In addition, to ensure that each NASA mission directorate is granted their allocated share of resources at any given time, mission directorate limits (called "shares") are also set on Pleiades and Columbia.

See **man pbs** for more information.

Job Accounting

Usage on the HECC machines at NAS, except for the front-end machines, is charged.

Starting May 1, 2011, the accounting unit is the Standard Billing Unit (SBU). The SBUs charged to a PBS job running on the compute node(s) is:

$$\text{SBU charged} = \text{Wall_Clock_Hours_Used} * \text{Number of MAUs} * \text{SBU Rate}$$

where the MAU represents the minimum allocatable unit of resources available through PBS. On Pleiades, an MAU is a node (with 8 cores for Harpertown and Nehalem-EP, 12 cores for Westmere, and 16 cores for Sandy Bridge in each node). On Columiba, an MAU has 4 cores. Charging is based on the number of MAUs allocated to a job, not how many cores are actually used during runtime. Once a user is allocated the resources, that user has exclusive access to those resources until the user's job completes or exceeds its requested wall-clock time.

The SBU rate for each of the NAS processors is outlined below:

Host	SBU Rate (per MAU)
Pleiades Sandy Bridge nodes	1.82
Pleiades Westmere nodes	1.00
Pleiades Nehalem-EP nodes	0.80
Pleiades Harpertown nodes	0.45
Columbia Itanium-2	0.18

In addition, charges on Columbia apply both to jobs that run successfully and those that are interrupted. Interrupted jobs are charged by taking the elapsed job time in hours, subtracting 1 hour, multiplying that by the number of MAUs used, and then deducting the resulting amount from the allocation. Users are encouraged to have their applications checkpoint roughly every hour.

In the near future, interruptions on Pleiades will be handled in a similar manner. For example:

You have a 24-hour job on Columbia that requires 16 MAUs (that is, 64 cores) that has run for 12 hours and then the system crashes. The accounting system will take the 12 hours, subtract 1 hour, and compute the SBUs (11 hours X 16 MAUs x 0.18 = 31.68 SBUs), which will then be subtracted from your allocation for your GID.

Job Accounting Utilities

Columbia Phase Out:

As of Feb. 8, 2013, the Columbia21 node has been taken offline as part of the Columbia phase out process. Columbia22-24 are still available. If your script requires a specific node, please make the appropriate changes in order to ensure the success of your job.

The job accounting utilities **acct_ytd** and **acct_query** can be used to obtain resource usage and charging information about your account, the accounts of other users on your project, and the project as a whole. Daily usage totals for each account are available for the current operational period.

acct_ytd

The **acct_ytd** command provides a year-to-date summary of accounting information for groups to which a user belongs. It will normally be accurate as of midnight the previous night, when accounting was last run.

A number of parameters can be used, but the simplest way is to type **acct_ytd** on a host without any parameters. This produces a line of output for each project you have access to on that host.

```
%acct_ytd
```

You can also specify the host group and/or a specific GID (for example, a0800):

```
%acct_ytd -cpleiades a0800 %acct_ytd -ccolumbia a0800
```

To find the allocations and usages of all your GIDs on all hosts, use the **-call** flag.

```
%acct_ytd -call
```

See **man acct_ytd** on Pleiades and Columbia for more information.

acct_query

The **acct_query** command searches and displays process-level billing records. This means that while totals over a period or for each day in a period are possible, you can also obtain detailed billing records for each process run in a period.

For example, to see all the SBU usage, beginning June 1, 2010, ending July 1, 2010, for all projects and on all hosts by user zsmith:

```
%acct_query -b06/01/10 -e07/01/10 -pall -call -uzsmith
```

To see the current SBU usage for the operational year 2010 (defined as May 1, 2010 to May 1, 2011 for most mission directorates) for all projects and on all hosts by user zsmith:

```
%acct_query -y10 -pall -call -uzsmith
```

Eligible hostnames include:

- columbia21
- columbia22
- columbia23
- columbia24
- pbs1
- pleiades (for Harpertown nodes)
- pleiades_N (for Nehalem nodes)
- pleiades_W (for Westmere nodes)
- pleiades_S (for Sandy Bridge nodes)

See **man acct_query** on Pleiades and Columbia for more information.

Multiple GIDs and Charging to a specific GID

Each approved project is assigned a project ID (GID). Members of a GID are authorized to use the resources allocated to that GID. For those users who have access to multiple GIDs, be aware that only one of those GIDs is considered your default.

Use the **groups** command to find which GIDs you are a member of. The following example shows that user zsmith is a member of the groups a0800, a0907, all, and e0720.

```
%groups zsmith
zsmith : a0800 a0907 all e0720
```

The first GID from the "groups" list should be your default GID. This can be verified through the **/etc/passwd** file. For example, the **/etc/passwd** file has an entry for user zsmith with the GID 20800 (which is the same as a0800, his default GID).

```
%grep zsmith /etc/passwd
zsmith:x:6666:20800:Z. Smith,,650-604-4444,:/u/zsmith:/bin/csh
```

When you use resources on the compute nodes through PBS jobs, SBUs are deducted from your default GID unless you specify otherwise. To charge resource usage to an alternative GID for a batch job, you can use the PBS flag **-W group_list=account** either in your script or on the **qsub** command line. For example:

```
#PBS -W group_list=a0907
or
```

```
%qsub -W group_list=a0907
```

Commonly Used PBS Commands

man pbs provides a list of all PBS commands. The four most commonly used PBS commands, **qsub**, **qstat**, **qdel**, and **qhold**, are briefly described below.

qsub

To submit a batch job to the specified queue using a script:

```
%qsub -q queue_name job_script
```

Common possibilities for *queue_name* at NAS include *normal*, *debug*, *long*, *devel*, and *low*. When *queue_name* is omitted, the job is routed to the default queue, which is the *normal* queue.

To submit an interactive PBS job:

```
%qsub -I -q queue_name -lresource_list
```

No *job_script* should be included when submitting an interactive PBS job.

The *resource_list* typically specifies the number of nodes, CPUs, amount of memory and walltime needed for this job. The following example shows a request for Pleiades with 2 nodes, 8 CPUs per node, and a walltime limit of 3 hours.

```
%qsub -I -lselect=2:ncpus=8,walltime=3:00:00
```

See **man pbs_resources** for more information on what resources can be specified. If **-lresource_list** is omitted, the default resources for the specified queue is used. When *queue_name* is omitted, the job is routed to the default queue, which is the *normal* queue.

qstat

To display queue information:

```
%qstat -Q queue_name
```

```
%qstat -q queue_name
```

```
$qstat -fQ queue_name
```

These commands display in different formats all the queue available on the systems, their constraints and status. The *queue_name* is optional.

To display job status using **qstat -:**

-a

Display all jobs in any status (running, queued, held)

-r

- n Display all running or suspended jobs
- i Display the execution hosts of the running jobs
- j Display all queued, held or waiting jobs
- u *user_name* Display jobs that belong to the specified user
- s Display any comment added by the administrator or scheduler. This option is typically used to find clues of why a job has not started running.
- f *job_id* Display detailed information about a specific job
- xf *job_id*, -xu *user_id* Display status informaton for finished jobs (within the past 7 days). This option is only available in newer version of PBS, which has been installed on Pleiades, but not on Columbia.

TIP: Some of these flags can be combined when checking the job status.

qdel

To delete a job:

```
%qdel job_id
```

qhold

To hold a job:

```
%qhold job_id
```

Only the job owner or a system administrator with "su" or "root" privilege can place a hold on a job. The hold can be released using the **qrls** command.

For more detailed information on each command, see their corresponding **man pages**.

Commonly Used QSUB Options in PBS Scripts or in the QSUB Command Line

The `qsub` options can be read from the PBS directives of a PBS *job_script* or from the `qsub` command line. For a complete list of available options, see **man qsub**. The more commonly used ones are listed below.

- S *shell_name*
Specifies the shell that interprets the job script
- V
Declares that all environment variables in the `qsub` command's environment are to be exported to the batch job
- v *variable_list*
Lists environment variables to be exported to the job
- q *queue_name*
Defines the destination of the job. The common possibilities for *queue_name* on Pleiades and Columbia include *normal*, *debug*, *long*, and *low*. In addition, there is a *devel* queue on Pleiades for development work.
- l *resource_list*
Specifies the resources that are required by the job and establishes a limit to the amount of resources that can be consumed. Commonly used resource items are *select*, *ncpus*, *walltime*, and *memory*. See **man pbs_resources** for a complete list of available resources.
- e *path*
Directs the standard error output produced by the request to the stated file path
- o *path*
Directs the standard output produced by the request to the stated file path.
- j *join*
Declares that the standard output and error streams of the job should be merged (joined). The values for *join* can be:
 - oe*: standard output and error streams are merged in the standard output file
 - eo*: standard error and output streams are merged in the standard error file
- m *mail_options*
Defines the set of conditions under which the execution server will send mail message about the job. See **man qsub** for a list of *mail_options*.
- N *name*
Declares a name for the job
- W *addl_attributes*
Allows for the specification of additional job attributes. The most common ones are:
 - W group_list=*g_list*** specifies the group the job runs under
 - W depend=afterany: *job_ID.server_name.nas.nasa.gov*** (for example, 12345.pbspl1.nas.nasa.gov) submits a job which is to be executed after *job_ID* has finished with any exit status
 - W depend=afterok: *job_ID.server_name.nas.nasa.gov*** (for example, 12345.pbspl1.nas.nasa.gov) submits a job which is to be executed after *job_ID* has finished with no errors

`-r y/n`

Declares whether the job is rerunnable

The top of a PBS *job_script* contains PBS directives, each of which begins with the string **#PBS**. Here is an example for use on Pleiades.

```
#PBS -S /bin/csh
#PBS -V
#PBS -q long
#PBS -lselect=2:ncpus=8:mpiprocs=4:model=har,walltime=24:00:00
#PBS -j oe
#PBS -o /nobackup/zsmith/my_pbs_output
#PBS -N my_job_name
#PBS -m e
#PBS -W group_list=a0907
#PBS -r n
```

The resources and/or attributes set using options to the **qsub** command line override those set in the directives in the PBS *job_script*.

New Features in PBS

Some of the new features relevant to users are listed below:

Use of the Shrink-to-Fit Feature (version 11.3)

The shrink-to-fit (STF) feature allows a user to specify a range of acceptable walltimes for a job, so that PBS can run the job sooner than it might otherwise. This feature is particularly helpful when scheduling jobs before an upcoming dedicated time.

For example, suppose your typical job requires 5 days of walltime. If there were less than 5 days left before the start of dedicated time, the job wouldn't run until after dedicated time. However, if you know that your job can do enough useful work running for 3 days or longer, you can submit it in the following way:

```
% qsub -l min_walltime=72:00:00,max_walltime=120:00:00 job_script
```

When PBS attempts to run your job it will initially look for a time slot of 5 days; but when no such time slot is found between now and the dedicated time, it will look for smaller and smaller time slots, down to the **min_walltime** of 3 days.

If you have an existing job that is still queued, you can use the **qalter** command to add these **min_walltime** and **max_walltime** attributes:

```
% qalter -l min_walltime=hh:mm:ss,max_walltime=hh:mm:ss jobid
```

Or change the walltime with the command:

```
% qalter -l walltime=hh:mm:ss jobid
```

Show the Processor Model (version 10.4)

Processor model (for example, Harpertown, Nehalem-EP, Westmere, and Sandy Bridge) can be displayed with:

```
%qstat -W o=+model
```

Show Job History (version 10.1)

Use the PBS **-x** option to obtain job history information, including the submission parameters, start/end time, resources used, etc., for jobs that finished execution, were deleted or are still running.

The job history for finished jobs is preserved for a specific duration. After the duration has expired, PBS deletes the job history information and it is no longer available. Currently, the duration is set to be **7 days** on Pleiades.

```
%qstat -fx job_id
```

Advance and Standing Reservations (version 9.2)

An advance reservation can be made 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 nodes every Wednesday from 5pm to 8pm, for the next month.

The reservation is made 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.

The following example shows the creation of an advance reservation asking for 1 node with 8 CPUs, a start time of 11:30 and a duration of 30 minutes.

```
%pbs_rsub -R 1130 -D 00:30:00 -l select=1:ncpus=8
```

A reservation can be deleted using the **pbs_rdel** command.

For more information, see **man pbs_rsub** and **man pbs_rdel**.

WARNING: Requests to use advance and standing reservations must be approved by NAS management. Only staff with special privilege can create the reservations for users.

Checkpointing and Restart

None of the NAS HEC systems has an automatic checkpoint capability made available by the operating system. For jobs that need lots of resources and/or long walltime, you should have a checkpoint/restart capability implemented in the source code or job script.

PBS automatically restarts unfinished jobs after system crashes. If you do not want PBS to restart your job, make sure to add the following in your PBS script:

```
#PBS -r n
```

PBS Environment Variables

There are a number of environment variables provided to the PBS job. Some are taken from the user's environment and carried with the job. Others are created by PBS. Still others can be explicitly created by the user for exclusive use by PBS jobs. All PBS-provided environment variable names start with the characters "PBS_". Some are then followed by a capital O ("PBS_O_") indicating that the variable is from the job's originating environment (that is, the user's).

The following lists a few useful PBS environment variables:

PBS_O_WORKDIR

Contains the name of the directory from which the user submitted the PBS job

PBS_O_PATH

Value of *PATH* from submission environment

PBS_JOBID

Contains the PBS job identifier

PBS_JOBDIR

Pathname of job-specific staging and execution directory

PBS_NODEFILE

Contains a list of vnodes assigned to the job

TMPDIR

The job-specific temporary directory for this job. Defaults to `/tmp/pbs.job_id` on the vnodes.

PBS Scheduling Policy

This article gives a simplified explanation of the PBS scheduling policy on Pleiades and Columbia.

PBS scheduling policies change frequently, in response to varying demands and workloads. The current policy (March 1, 2011), simplified, states that jobs are sorted in the following order: current mission directorate CPU use, job priority, queue priority, and job size (wide jobs first).

In each scheduling cycle, PBS examines the jobs in sorted order, starting a job if it can. If the job cannot be started immediately, it is either scheduled around or simply bypassed for this cycle.

There are numerous reasons why jobs won't start, such as:

- The queue is at its running job limit
- You are at your running job limit
- The queue is at its CPU limit
- The mission directorate is at its CPU share limit and the job cannot borrow from another mission
- Not enough CPUs are available

Notice that a high-priority job might be blocked by some limit, while a lower priority job, from a different user or asking for fewer resources, might not be blocked.

If your job is waiting in the queue, use the following commands to get some information about why it has not started running.

```
pfe20% qstat -s jobid
or
pfe20% qstat -f jobid | grep -i comment
```

On Pleiades, output from the following command shows the amount of resources (broken down into Harpertown, Nehalem, Westmere, and Sandy Bridge processors) used and borrowed by each mission directorate, and the resources each mission is waiting for:

```
pfe20% /u/scicon/tools/bin/qs
```

The following command provides the order of jobs that PBS schedules to start at the current scheduling cycle. It also provides information regarding processor type(s), mission, and job priority:

```
pfe20% qstat -W o=+model,mission,pri -i
```

The policy described above could result in a large, high-priority job being blocked forever by a steady stream of smaller, low-priority jobs. To prevent jobs from languishing in the queues for an indefinite time, PBS reserves resources for the top N jobs (currently, N is 4), and doesn't allow lower priority jobs start if they would delay the start time of one of the top job

("backfilling"). Additional details are given below.

PBS Sorting Order

Mission Shares

Each NASA mission directorate is allocated a certain percentage of the CPUs in the system. (See [Mission Shares Policy on Pleiades](#) .) A job cannot start if that action would cause the mission to exceed its share, unless another mission is using less than its share and has no jobs waiting. In this case, the high-use mission can "borrow" CPUs from the lower-use mission for up to a specified time (currently, `max_borrow` is 4 hours).

So , if the job itself needs less than `max_borrow` hours to run, or if a sufficient number of other jobs from the high-use mission will finish within `max_borrow` hours to get back under its mission share, then the job can borrow CPUs.

When jobs are sorted, jobs from missions using less of their share are picked before jobs from missions using more of their share.

Job Priority

Job priority has three components. First is the native priority (the `-p` parameter to `qsub` or `qalter`). Added to that is the queue priority. If the native priority is 0, then a further adjustment is made based on how long the job has been waiting for resources. Waiting jobs get a "boost" of up to 20 priority points, depending on how long they have been waiting and which queue they are in.

This treatment is modified for queues assigned to the Human Exploration and Operations Mission Directorate (HEOMD). For those queues, job priority is set by a separate set of policies controlled by HEOMD management.

Queue priority

Some queues are given higher or lower priorities than the default (run `qstat -Q` to get current values). Note that because the mission share is the most significant sort criterion, job and queue priorities have little effect mission-to-mission.

Job Size

Jobs asking for more nodes are favored over jobs asking for fewer. The reasoning is that, while it is easier for narrow jobs to fill in gaps in the schedule, wide jobs need help

collecting enough CPUs to start.

Backfilling

As mentioned above, when PBS cannot start a job immediately, if it is one of the first N such jobs, PBS sets aside resources for the job before examining other jobs. That is, PBS looks at the currently running jobs to see when they will finish (using the wall-time estimates). From those finish times, PBS decides when enough resources (such as CPUs, memory, mission share, and job limits) will become available to run the top job.

PBS then creates a virtual reservation for those resources at that time. Now, when PBS looks at other jobs to see if they can start immediately, it also checks whether starting the job would collide with one of these reservations. Only if there are no collisions will PBS start the lower priority jobs.

This description applies to both Pleiades and Columbia, although the specific queues, priorities, mission percentages, and other elements differ between the two systems.

PBS exit codes

The PBS exit value of a job may fall in one of four ranges:

- $X = 0$ (= JOB_EXEC_OK)

This is a PBS special return value indicating that the job executed successfully

- $X < 0$

This is a PBS special return value indicating that the job could not be executed.

These negative values are:

- ◆ -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 exec failed, do retry
- ◆ -4 = JOB_EXEC_INITABT : Job aborted on MOM initialization
- ◆ -5 = JOB_EXEC_INITRST : Job aborted on MOM initialization, checkpoint, no migrate
- ◆ -6 = JOB_EXEC_INITRMG : Job aborted on MOM initialization, checkpoint, ok migrate
- ◆ -7 = JOB_EXEC_BADRESRT : Job restart failed
- ◆ -8 = JOB_EXEC_GLOBUS_INIT_RETRY : Initialization of Globus job failed. Do retry.
- ◆ -9 = JOB_EXEC_GLOBUS_INIT_FAIL : Initialization of Globus job failed. Do not retry.
- ◆ -10 = JOB_EXEC_FAILUID : Invalid UID/GID for job
- ◆ -11 = JOB_EXEC_RERUN : Job was 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

- $0 \leq X < 128$ (or 256 depending on the system)

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).

- $X \geq 128$ (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 ($137 \% 128 = 9$).

Front-End Usage Guidelines

Pleiades Front-End Usage Guidelines

Summary: Use the Pleiades front-end systems (PFEs) and the bridge nodes for file editing, compiling, short debugging/testing sessions, and batch job submissions.

The PFEs and the bridge nodes are the front-end systems to Pleiades. They provide an environment that allows you to get quick turnaround while performing file editing, file transferring, compiling, short debugging/testing sessions, and batch job submission via PBS to a subset of the Pleiades compute nodes.

WARNING: The new Pleiades front-ends (pfe[20-27]) use the Intel Sandy Bridge processors. If you use a PGI compiler to build your executable, be aware that by default the executable is optimized for Sandy Bridge and will not necessarily execute on Harpertown, Nehalem-EP, or Westmere processors. To generate a single executable that will work on all Pleiades processor types, use the option

`-tp=penryn-64, nehalem-64, sandybridge-64` during compilation with PGI compilers. See [PGI Compilers and Tools](#) for more information.

You cannot "ssh" to the compute nodes except for the subset of nodes your PBS job is running on.

The bridge nodes are recommended for the following functions:

Pre- and/or Post-Processing

The large amount of memory on the bridge nodes allows pre- and post-processing applications such as [Tecplot](#), [IDL](#), and [Matlab](#) to run faster than on the PFEs. Note that the bridge nodes have the same software as the PFEs. For a list of available applications, run the command `module avail`.

File Transfers Between Pleiades and Columbia

Both the Pleiades Lustre filesystems `/nobackup` and the Columbia CXFS filesystems `/nobackup` are mounted on the bridge nodes. To copy files between the Pleiades Lustre and Columbia CXFS filesystems, log into a bridge node and use the `cp`, `mcp`, or `shiftpc` command to perform the transfer.

File Transfers to Mass Storage

The Pleiades /nobackup filesystems are mounted on Lou2. Thus, the easiest way to transfer files between Pleiades and Lou2 is to initiate a command such as **cp**, **mcp**, **tar**, or **shifc** on Lou2. For example:

```
lou% mcp /nobackup/username/foo $HOME
```

If you initiate the transfer on Pleiades, the commands **scp**, **bbftp**, **bbscp**, and **shifc** are available to do the transfers between a Pleiades front-end or bridge node and Lou. Since **bbscp** uses almost the same syntax as **scp**, but performs faster than **scp**, we recommend using **bbscp** in cases where you do not require the data to be encrypted. For very large file transfers, we recommend the Shift utility, developed at NAS.

See also [File Transfer Overview](#), and [File Transfer Commands](#).

File transfers from the Pleiades compute nodes to Lou must go through one of the PFEs or bridge nodes first. See [Streamlining File Transfers from the Pleiades Compute Nodes to Lou](#) for more information.

When sending data to Lou, keep your largest individual file size under 1 TB, as large files will occupy all of the tape drives, preventing other file restores and backups.

Additional Restrictions on Front-end Systems

- No MPI (Message Passing Interface) jobs are allowed to run on the PFEs or the bridge nodes
- A job on bridge[1-2] should not use more than 56 GB; when it does, a courtesy email is sent to the owner of the job
- A job on bridge[3-4] should not use more than 192 GB; when it does, a courtesy email is sent to the owner of the job

Before starting a large-memory session, it is a good idea to check to make sure there is enough memory available. You can run the command **top**, hit "M", and check under the "RES" column for other large memory applications that may be running.

Columbia Front-End Usage Guidelines

The front-end system, cfe2, provide an environment that allows users to get quick turnaround while performing the following: file editing, file management, short debugging and testing sessions, and batch job submission to the compute systems.

Running long and/or large (in terms of memory and/or number of processors) debugging or production jobs interactively or in the background of cfe2 is considered to be inconsiderate behavior to the rest of the user community. If you need help submitting such jobs to the batch systems, please contact a the Control Room at (650) 604-4444 or (800) 331-USER or send e-mail to: support@nas.nasa.gov

Jobs that cause significant impact on the system load of the Columbia front-end machine (cfe2) are candidates for removal in order to bring the front-end systems back to a normal and smooth environment for all users. A cron job regularly monitors the system load and determines if job removal is necessary. The criteria for job removal are described below. Owners of any removed jobs will receive a notification e-mail.

1. To be eligible for removal, the number of processors a front-end interactive job uses can be one (1) or more. Exceptions to this are those programs, utilities, etc. common to users and/or NASA missions that are listed in an "exception file". Examples of these would be: **bash**, **cp**, **cs**, **em**, **gz**, **rs**, **sc**, **sf**, **sh**, **ss**, **ta**, and **tc**. Users can submit program names to be added to this exception file by mailing requests to: support@nas.nasa.gov.
2. For qualifying processes, the CPU time usage of each process in a job has, on the average, exceeded a threshold defined as: (20 min x 8 / number of processes for the job). That is, a baseline for removal is a job with 8 processors running for more than 20 minutes. The maximum amount of time allowed for each processor in a job is scaled using the formula: 20 min x 8 cpu / number-of-processes. Therefore, the following variations are possible:
 - ◆ 160 minutes = (20 * 8) / 1 cpu
 - ◆ 80 minutes = (20 * 8) / 2 cpu
 - ◆ 40 minutes = (20 * 8) / 4 cpu
 - ◆ 20 minutes = (20 * 8) / 8 cpu
 - ◆ 10 minutes = (20 * 8) / 16 cpu
 - ◆ 5 minutes = (20 * 8) / 32 cpu
 - ◆ 2.5 minutes = (20 * 8) / 64 cpu

The conditions of removal are subject to change, when necessary.

PBS on Pleiades

Overview

Overview

On Pleiades, PBS (version 11.2) is used to manage batch jobs that run on the compute nodes (4 different processor types, 11,776 nodes, and 126,720 cores in total). PBS features that are common to all NAS systems are described in other articles. Read the following articles for Pleiades-specific PBS information:

- [Queue Structure](#)
- [Resource Request Examples](#)
- [Default Variables Set by PBS](#)
- [Sample PBS Scripts](#)

Queue Structure

You should be aware of the PBS queue structure to help with batch job management. To find the maximum and default NCPUS (number of CPUs), the maximum and default wall-time, the priority of the queue, and whether the queue is disabled or stopped, use the command:

```
% qstat -Q
```

This command also provides statistics of jobs in the states of queued (Q), held (H), running (R), or exiting (E).

Note that the queue structure will change from time to time. Below is a snapshot of the output from this command on August 21, 2012.

```
%qstat -Q
Queue      Ncpus/      Time/      State counts
name       max/def     max/def    jm T/_Q/_H/W/_R/E/B pr  Info
=====
alphanst   --/ --     120:00/ 01:00 -- 0/_0/_0/0/_0/0/0 0 disabled stopped
dpr        --/ 8       --/ 00:10 -- 0/_0/_0/0/_0/0/0 0 disabled
smd2       2048/ 8     12:00/ 01:00 -- 0/_0/_0/0/_0/0/0 16 disabled
diags      --/ --     120:00/ 04:00 -- 0/_0/_0/0/_0/0/0 0
wide       --/ --     120:00/ 01:00 -- 0/_0/_0/0/_3/0/0 101 disabled
rtf        --/ 8       --/ 01:00 -- 0/_0/_0/0/_0/0/0 65 disabled
somd_spl   --/ 8     240:00/ 01:00 -- 0/_0/_0/0/_0/0/0 25 disabled
heomd_spl  --/ 8     120:00/ 01:00 -- 0/_0/_0/0/_2/0/0 45
smd1       --/ 8     240:00/   -- -- 0/_0/_0/0/_0/0/0 16
vlong      8192/ 8     384:00/120:00 -- 0/_1/_0/0/_1/0/0 0
arnd_spl   4900/ 8     120:00/ 01:00 -- 0/_0/_0/0/_0/0/0 15 disabled
ded_time   --/ --     --/ 01:00 -- 0/_0/_1/0/_0/0/0 0
idle       --/ --     --/   -- -- 0/_0/_0/0/_0/0/0 0 disabled
testing    --/ --     --/ 00:30 -- 0/_0/_0/0/_0/0/0 0
sls_aero1  --/ 8     48:00/ 01:00 -- 0/_60/_0/0/_56/0/0 45
kepler     --/ 8     120:00/ 01:00 -- 0/_0/_0/0/_0/0/0 101
sls_aero2  408/ 8     144:00/ 01:00 -- 0/_0/_0/0/_0/0/0 45 disabled
low      --/ 8     04:00/ 00:30 -- 0/_0/_0/0/_0/0/0 -10
gpu_long_free --/ 24    24:00/ 01:00 -- 0/_0/_0/0/_0/0/0 0
normal   --/ 8     08:00/ 01:00 -- 0/_77/17/0/104/0/0 0
long     8192/ 8     120:00/ 01:00 -- 0/156/_3/0/338/0/0 0
debug    1025/ 8     02:00/ 00:30 -- 0/_6/_0/0/_7/0/0 15
route      --/ 8       --/   -- -- 0/_0/_2/0/_0/0/0 0
devel    4800/ 1     02:00/   -- -- 0/_22/_0/0/_9/0/0 60
gpu      --/ 8     08:00/ 01:00 -- 0/_1/_0/0/_0/0/0 0
```

To view more information about each queue, use:

```
% qstat -fQ queue_name
```

In the output of this command, you will find additional information such as:

```
acl_groups
```

Lists all GIDs that are allowed to run in the queue. For the **normal**, **debug**, **long**, **low**, and **devel** queues, all GIDs should be included. Special queues, such as **esmd_spl**, **arnd_spl**, **somd_spl**, **clv_spl**, etc., typically allow a few GIDs only.

default_chunk.model

Specifies the default processor model, if you do not specify the processor model yourself. All queues, except **devel** and **gpu**, default to using nodes with Harpertown model processors. See [Pleiades devel Queue](#) for the defaults for the **devel** queue and [GPU Basics](#) for more information on the **gpu** queue.

resources_min.ncpus

If defined, specifies the minimum NCPUs required for the queue. The **wide** queue requires using a minimum of 1024 CPUs.

max_run

If defined, specifies the maximum number of jobs each user is allowed to run in the queue. For the **devel** queue, the maximum number is set at 10. For the **debug** queue, it is set at 2.

TIP: To request using the Nehalem-EP, Westmere, or Sandy Bridge nodes, use the attributes **model=neh**, **model=wes**, or **model=san** in your **resource_list**. To explicitly request Harpertown nodes, use **model=har**. You can apply the model attribute to any queue.

For example:

```
#PBS -q long
#PBS -l select=1:ncpus=12:model=wes
```

Mission Shares Policy on Pleiades

Mission Directorate shares have been implemented on Pleiades since Feb. 10, 2009. Implementing shares guarantees that each Mission Directorate gets its fair share of resources.

The share to which a job is assigned is based on the GID used by the job. Once all the cores within a Mission Directorate's share have been assigned, other jobs assigned to that share must wait, even if cores are available in a different Mission Directorate's share, with the following exception:

When a Mission Directorate is not using all of its cores, other Mission Directorates can borrow those cores, but only for jobs that will finish within 4 hours. When part of the resource is unavailable, the total number of cores decreases, and each Mission Directorate loses a proportionate number of cores.

You can display the share distribution by adding the `-W shares=` option to the `qstat` command. For example:

```
%qstat -W shares=
```

Group	Share%	Use%	Share	Exempt	Use	Avail	Borrowed	Ratio	Waiting
Overall	100	0	159748	0	960	158788	0	0.01	960
ARMD	24	18	38109	0	29680	8429	0	0.78	22512
HEOMD	23	21	36521	0	34312	2209	0	0.94	28416
SMD	51	50	80981	0	80968	13	0	1.00	113920
NAS	2	0	3175	0	0	3175	0	0.00	20240

Mission shares are calculated by combining the mission's HECC share of the shared assets combined with the mission-specific assets. The mission shares on Oct 3, 2011 are shown in the second column of the above display. The amount of resources used and borrowed by each mission and resources each mission is waiting for are also displayed.

An in-house utility, `qs`, provide similar information with details that break the resources into the Harpertown, Nehalem-EP, Westmere, and Sandy Bridge processor types and is available at `/u/scicon/tools/bin/qs`.

The `-h` option of `qs` provides instructions on how to use it:

```
% /u/scicon/tools/bin/qs -h
```

```
usage: qs [-u] [-n N] [-b] [-p] [-d] [-r] [-f M,N] [-q N] [-t] [-v] [-h] [--file f]
        -u          : show used resources only; don't show queued jobs
        -n N        : show time remaining before N nodes are free
        -b          : order segments in bars to help understand borrowing
        -p          : plain output: i.e. no colors or highlights
```


Resources Request Examples

Since Pleiades consists of four different processor types (Harpertown, Nehalem-EP, Westmere, and Sandy Bridge), you will benefit from keeping the following in mind when requesting PBS resources for your job:

Charging on the usage of the four Pleiades processor types is based on a common Standard Billing Unit which is on a per-node basis. The SBU rate for each of the Pleiades processor types is:

Processor Type	SBU Rate (per node)
Sandy Bridge	1.82 (16 cores in a node)
Westmere	1.0 (12 cores in a node)
Nehalem-EP	0.80 (8 cores in a node)
Harpertown	0.45 (8 cores in a node)

The actual amount of memory per node through PBS is slightly less than 7.6 GB per node for Harpertown, 22.5 GB per node for Nehalem-EP and Westmere, and about 31 GB per node for Sandy Bridge.

For the **normal**, **long**, and **debug** queue, use the **model=[har, neh, wes, san]** attribute to request the processor type(s) for your job. If the processor type is not specified in your PBS resource list, the job is routed to use the default processor type, Harpertown, for most queues. For the **devel** queue, see [Pleiades devel queue](#) for more information. For the **gpu** queue, see [GPU Basics](#).

Example 1

Here are some examples of requesting certain processor models for a 128-process job:

```
#PBS -l select=16:ncpus=8:model=har
#to run all 8 cores on each of 16 Harpertown nodes

#PBS -l select=32:ncpus=4:model=har
#to run on only 4 cores on each of 32 Harpertown nodes
#(note: will be charged for 32 nodes = 256 cores)

#PBS -l select=16:ncpus=8:model=neh
#to run all 8 cores on each of 16 Nehalem-EP nodes

#PBS -l select=11:ncpus=12:model=wes
#to run all 12 cores on each of 11 Westmere nodes
#(4 cores in 11th node will go unused)

#PBS -l select=8:ncpus=16:model=san
#to run all 16 cores on each of 8 Sandy Bridge nodes
```

Note that you can specify both the queue type (`-q normal, debug, long, low`) and the processor type (`-l model=har, neh, wes, san`). For example:

```
#PBS -q normal
#PBS -l select=16:ncpus=8:model=neh
```

If your application can run on any of the four processor types, you may want to submit your job to a processor type that has more nodes unoccupied by other running jobs. Doing this can possibly reduce the wait time of your job. The script `node_stats.sh` provides information about the total, used, and free nodes for each processor type. For example:

```
% /u/scicon/tools/bin/node_stats.sh
```

```
Pleiades Nodes Total: 11240
Pleiades Nodes Used : 10485
Pleiades Nodes Free : 755
```

```
Harpertown   Total: 4063, Used: 4015, Free: 48
Nehalem      Total: 1253, Used: 847, Free: 406
Westmere     Total: 3957, Used: 3916, Free: 41
SandyBridge  Total: 1288, Used: 1128, Free: 160
GPU nodes    Total: 56, Used: 2, Free: 54
Devel queue  Total: 623, Used: 577, Free: 46
```

```
Currently queued jobs are requesting: 8259 Harpertown, 2142 Nehalem, 9633 Westmere,
```

TIP: Add `/u/scicon/tools/bin` to your path in `.cshrc` or other shell start-up scripts to avoid having to type in the complete path for this tool.

For each job, you can also identify which processor models are used by looking at the "Model" field of the output of the command:

```
% qstat -a -W o=+model
```

Example 2

The Harpertown nodes in rack 32 have 16 GB memory/node instead of 8 GB per node.

This example shows a request of 2 nodes with bigmem in rack 32.

```
#PBS -l select=2:ncpus=8:bigmem=true:model=har
```

Example 3

For a multi-node PBS job, the NCPUs used in each node can be different. This is useful for jobs that need more memory for some processes, but less for other processes. Resource requests can be done in "chunks" for a job with varying NCPUs per node.

This example shows a request of two resource chunks. In the first chunk, 1 CPU in 1 node, and in the second chunk, 8 CPUs in each of three other nodes are requested:

```
#PBS -l select=1:ncpus=1+3:ncpus=8
```

Example 4

A PBS job can run across different processor types. This can be useful in two scenarios:

- When you cannot find enough free nodes within one model for your job
- When some of your processes need more memory while others need much less

This can be accomplished by specifying the resources in chunks, with one chunk requesting one processor type and another chunk requestings a different processor type.

Here is an example of how to request 1 Westmere node (which provides 24 GB per node) and 3 Harpertown nodes (which provides 8 GB per node), under either of the following circumstances:

```
#PBS -lplace=scatter:excl:group=model
```

```
#PBS -lselect=1:ncpus=12:mpiprocs=12:model=wes+3:ncpus=8:mpiprocs=8:model=har
```

Default Variables Set by PBS

You can use the `env` command--either in a PBS script or on the command line of a PBS interactive session--to find out what environment variables are set within a PBS job. In addition to the PBS_xxxx variables, the following two are useful to know:

NCPUS

Defaults to number of CPUs that you requested for the node.

OMP_NUM_THREADS

Defaults to 1 unless you explicitly set it to a different number. If your PBS job runs an OpenMP or MPI/OpenMP application, this variable sets the number of threads in the parallel region.

FORT_BUFFERED

Defaults to 1. Setting this variable to 1 enables records to be accumulated in the buffer and flushed to disk later.

Sample PBS Script for Pleiades

```
#PBS -S /bin/csh
#PBS -N cfd
# This example uses the Harpertown nodes
# User job can access ~7.6 GB of memory per Harpertown node.
# A memory intensive job that needs more than ~0.9 GB
# per process should use less than 8 cores per node
# to allow more memory per MPI process. This example
# asks for 64 nodes and 4 MPI processes per node.
# This request implies 64x4 = 256 MPI processes for the job.
#PBS -l select=64:ncpus=8:mpiprocs=4:model=har
#PBS -l walltime=4:00:00
#PBS -j oe
#PBS -W group_list=a0801
#PBS -m e

# Currently, there is no default compiler and MPI library set.
# You should load in the version you want.
# Currently, MVAPICH or SGI's MPT are available in 64-bit only,
# you should use a 64-bit version of the compiler.

module load comp-intel/11.1.072
module load mpi-sgi/mpt.2.04.10789

# By default, PBS executes your job from your home directory.
# However, you can use the environment variable
# PBS_O_WORKDIR to change to the directory where
# you submitted your job.

cd $PBS_O_WORKDIR

# use of dplace to pin processes to processors may improve performance
# Here you request to pin processes to processors 2, 3, 6, 7 of each node.
# This helps for using the Harpertown nodes, but not for Nehalem-EP or
# Westmere-EP nodes

# The resource request of select=64 and mpiprocs=4 implies
# that you want to have 256 MPI processes in total.
# If this is correct, you can omit the -np 256 for mpiexec
# that you might have used before.

mpiexec dplace -s1 -c2,3,6,7 ./grinder < run_input > output

# It is a good practice to write stderr and stdout to a file (ex: output)
# Otherwise, they will be written to the PBS stderr and stdout in /PBS/spool,
# which has limited amount of space. When /PBS/spool is filled up, any job
# that tries to write to /PBS/spool will die.

# -end of script-
```

Pleiades devel Queue

NAS provides a special **devel** queue that provides faster turnaround when doing development work.

Currently, 512 Westmere nodes and 144 Sandy Bridge nodes are reserved for the Pleiades **devel** queue, 24x7. The maximum walltime allowed is 2:00:00 and the maximum NCPUS is 4800. Each user is allowed to have only one job running in the **devel** queue at any one time.

To specify the Westmere processor type to be used for your job in the **devel** queue, do:

```
#PBS -l select=xx:ncpus=yy:model=wes
```

If you want to use the Sandy Bridge nodes, do:

```
#PBS -l select=xx:ncpus=yy:model=san
```

To submit your job to the **devel** queue, do:

```
pfe20% qsub -q devel job_script  
1234.pbspl1.nas.nasa.gov
```

To check the status of your job submitted to the **devel** queue, do:

```
pfe20% qstat devel -u your_username
```

PBS on Columbia

Overview

Columbia Phase Out:

As of Feb. 8, 2013, the Columbia21 node has been taken offline as part of the [Columbia phase out process](#). Columbia22-24 are still available. If your script requires a specific node, please make the appropriate changes in order to ensure the success of your job.

On Columbia, PBS (version 10.4) is used to manage batch jobs that run on the four compute systems (Columbia21-24). PBS features that are common to all NAS systems are described in other articles. Read the following articles for Columbia-specific PBS information:

- [Resource Request Examples](#)
- [Default Variables Set by PBS](#)
- [Sample PBS Scripts](#)

Resources Request Examples

Columbia Phase Out:

As of Feb. 8, 2013, the Columbia21 node has been taken offline as part of the [Columbia phase out process](#). Columbia22-24 are still available. If your script requires a specific node, please make the appropriate changes in order to ensure the success of your job.

All of the Columbia compute engines, Columbia21-24, are single-system image Altix 4700 systems:

```
Columbia21 (508 CPUs total, 1.8 GB memory/CPU through PBS)
Columbia22 (2044 CPUs total, 1.8 GB memory/CPU through PBS)
Columbia23 (1020 CPUs total, 1.8 GB memory/CPU through PBS)
Columbia23 (1020 CPUs total, 1.8 GB memory/CPU through PBS)
```

Here are a few examples of requesting resources on Columbia:

Example 1

If your job needs fewer than 508 CPUs and you do not care which Columbia system to run your job on, simply use `ncpus` to specify the number of CPUs that you want for your job. For example:

```
#PBS -l ncpus=256
```

Example 2

If you specify both the `ncpus` and `mem` for your job, PBS will make sure that your job is allocated enough resources to satisfy both `ncpus` and `mem`. For example, if you request 4 CPUs and 14 GB of memory, your job will be allocated 8 CPUs and 14.4 GB because the amount of memory associated with 4 CPUs is not enough to satisfy your memory request.

```
#PBS -l ncpus=4,mem=14GB
```

Example 3

If you want your job to run on a specific Columbia machine, for example, Columbia22 with 256 CPUs, use:

```
#PBS -l select=host=columbia22:ncpus=256
```

Note that the `ncpus` request must appear with the `select=host` request and must not be present as a separate request either on the `qsub` command line or in the PBS script.

Example 4

If you ever need to run a job across two Columbia systems, for example, 508 CPUs on one Columbia and another 508 CPUs on another, use:

```
#PBS -l select=2:ncpus=508,place=scatter
```

Default Variables Set by PBS

You can use the `env` command--either in a PBS script or from the command line of an interactive PBS session--to find out what environment variables are set within a PBS job. In addition to the PBS_XXX variables, the following ones are useful to know:

NCPUS

Defaults to number of CPUs that you requested.

OMP_NUM_THREADS

Defaults to 1 unless you explicitly set it to a different number. If your PBS job runs an OpenMP or MPI/OpenMP application, this variable sets the number of threads in the parallel region.

OMP_DYNAMIC

Defaults to *false*. If your PBS job runs an OpenMP application, this disables dynamic adjustment of the number of threads available for execution of parallel regions.

MPI_DSM_DISTRIBUTE

Defaults to *true*. If your PBS job runs an MPI application, this ensures that each MPI process gets a unique CPU and physical memory on the node with which that CPU is associated.

FORT_BUFFERED

Defaults to 1. Setting this variable to 1 enables records to be accumulated in the buffer and flushed to disk later.

Sample PBS Script for Columbia

```
#PBS -S /bin/csh
#PBS -N cfd
#PBS -l ncpus=4
#PBS -l mem=7776MB
#PBS -l walltime=4:00:00
#PBS -j oe
#PBS -W group_list=g12345
#PBS -m e

# By default, PBS executes your job from your home directory.
# However, you can use the environment variable
# PBS_O_WORKDIR to change to the directory where
# you submitted your job.

cd $PBS_O_WORKDIR

# For MPI jobs, there is an SGI MPT module loaded by default, unless you
# modify your shell start up script to unload it or switch to a different
# version. You can use either mpiexec or mpirun to start your job.

mpiexec -np 4 ./a.out < input > output

# It is a good practice to write stderr and stdout to a file (ex: output)
# Otherwise, they will be written to the PBS stderr and stdout in /PBS/spool
# which has limited amount of space. When /PBS/spool is filled up, any job
# that tries to write to /PBS/spool will die.

# -end of script-
```

Troubleshooting PBS Jobs

Common Reasons for Being Unable to Submit Jobs

There are several common reasons why you might not be able to successfully submit a job to PBS:

Resource Request Exceeds Resource Limits

If you get the following message after submitting a PBS job:

```
"qsub: Job exceeds queue resource limits"
```

Reduce your resource request to below the limit or use a different queue.

AUID or GID not Authorized to Use a Specific Queue

If you get the following message after submitting a PBS job:

```
"qsub: Unauthorized Request"
```

It is possible that you tried submitting to a queue which is accessible only to certain groups or users. You can check the `qstat -fQ` output and see if there is an `acl_groups` or a `acl_users` list. If your group or username is not in the lists, you will have to submit to a different queue.

If your GID has no allocation left, you will also get this error. See the section 'Not Enough or No Allocation Left' below for more information.

AUID Not Authorized to Use a Specific GID

If you get the following message after submitting a PBS job:

```
"qsub: Bad GID for job execution"
```

It is possible that your AUID has not been added to use allocations under a specific GID. Please consult with the principal investigator of that GID and ask him/her to submit a request to support@nas.nasa.gov to add your AUID under that GID.

Queue is Disabled

If you get the following message after submitting a PBS job:

```
"qsub: Queue is not enabled"
```

You should submit to a different queue which is enabled.

Not Enough or No Allocation Left

An automated script is used to check if a GID is over its allocation everyday. If it does, that GID is removed from PBS access control list and users of that GID will not be able to submit jobs.

Users can check the amount of allocations remaining using the `acct_ytd` command. In addition, if you see in your PBS output file some message regarding your GID allocation usage is near its limit or is already over, ask your PI to request for more allocation.

Once the request for more allocation is approved and added to your account, an automatic hourly script will add your GID back to the PBS access control list.

Common Reasons Why Jobs Won't Start

If your job does not run after it has been successfully submitted, it might be due to one of the following reasons:

- The job is waiting for resources
- Your mission share has run out
- The system is going into dedicated time
- Scheduling is turned off
- The job has been placed on hold
- Your home filesystem or default /nobackup filesystem is down

You can often find out why a job does not start by running `tracejob jobid` on the PBS server `pbsp11`, which prints log messages for a PBS job, or `qstat -as jobid` on any Pleiades front-end systems, which displays all information about a job.

The following sections provide more details about each scenario, along with troubleshooting steps.

The Job is Waiting for Resources

Your job might be waiting for resources for one of the following reasons:

- All resources are busy running jobs, and no other jobs can be started until resources become available again
- There is a higher-priority job that needs more resources than are currently available, so PBS is draining the system and not running any new jobs in order to accommodate the high-priority job
- Users have submitted too many jobs at once (for example, more than 100), so the PBS scheduler is busy sorting jobs and cannot start new jobs effectively
- If you requested a specific node or group of nodes to run your job, it might wait in the queue longer than if nodes were not specified

To view job status and events, run the `tracejob` utility on `pbsp11`. For example:

```
pbsp11 $ tracejob 234567
Job: 234567.pbsp11.nas.nasa.gov

06/15/2012 00:23:55 S Job Modified at request of Scheduler@pbsp11.nas.nasa.gov
06/15/2012 00:23:55 L No available resources on nodes
06/15/2012 00:38:47 S Job Modified at request of Scheduler@pbsp11.nas.nasa.gov
06/15/2012 00:50:30 S Job Modified at request of Scheduler@pbsp11.nas.nasa.gov
06/15/2012 01:51:21 S Job Modified at request of Scheduler@pbsp11.nas.nasa.gov
06/15/2012 01:55:38 S Job Modified at request of Scheduler@pbsp11.nas.nasa.gov
06/15/2012 02:16:44 S Job Modified at request of Scheduler@pbsp11.nas.nasa.gov
06/15/2012 07:39:48 L Considering job to run
06/15/2012 07:39:48 L Job is requesting an exclusive node and node is in use
```

TIP: If the scheduler has not yet reviewed the job, no information will be available and the `tracejob` utility will not provide any output.

If your job requests an exclusive node and that node is in use, you can wait for the requested node, request a different node, or submit the job again without requesting a specific node.

To view the current node usage for each processor type, run the `node_states.sh` script. For example:

```
/u/scicon/tools/bin/node_stats.sh
```

```
Pleiades Nodes Total: 12457
Pleiades Nodes Used : 11675
Pleiades Nodes Free : 782

Harpertown   Total: 4083, Used: 3948, Free: 135
Nehalem      Total: 1279, Used: 1246, Free: 33
Westmere     Total: 4662, Used: 4436, Free: 226
SandyBridge  Total: 1723, Used: 1513, Free: 210
GPU nodes    Total: 62, Used: 2, Free: 60
Devel wes    Total: 504, Used: 418, Free: 86
Devel san    Total: 144, Used: 112, Free: 32
```

```
Currently queued jobs are requesting: 1734 Harpertown, 1502 Nehalem, 5219 Westmere,
```

TIP: Add `/u/scicon/tools/bin` to your path in `.cshrc` or other shell start-up scripts to avoid having to type in the complete path for this tool.

Your Mission Share Has Run Out

If all of the cores within your mission directorate's share have been assigned, or if the new job would exceed your mission share, your job will not run. If resources appear to be available, they belong to other missions.

To view all information about your job, run `qstat -as jobid`. In the following sample output, a comment indicates that the job would exceed the mission share:

```
Qstat as 778574
JobID          User      Queue   Jobname  CPUs  wallt  Ss  wallt  Eff  wallt
-----
778574.pbsp13  msmith3  normal  my_GC    12    04:00  Q   07:06  --   04:00
Job would exceed mission CPU share
```

To view the share distribution among all mission directorates, run `qstat -W shares`. For example:

```
pfe20 $ qstat -W shares
Group  Share% Use%  Share Exempt  Use  Avail Borrowed Ratio Waiting
-----
```

Overall	100	0	167456	0	0	167456	0	0.00	0
ARMD	25	20	41864	0	34480	7384	0	0.82	11744
HEOMD	23	17	38514	0	29632	8882	0	0.77	8304
SMD	50	45	83728	2912	76728	7000	0	0.92	129376
NAS	2	0	3349	0	384	2965	0	0.11	32

If your job exceeds your mission share, you might be able to borrow nodes from other mission directorates. To borrow nodes, your job must not request a wall-clock time that is too long (more than 4 hours on Pleiades). See [Mission Shares Policy on Pleiades](#) for more information.

The System is Going into Dedicated Time

When dedicated time (DED) is scheduled for hardware and/or software work, the PBS scheduler will not start a job if its projected end-time runs past the beginning of the DED time.

If you can reduce the requested wall-clock time so that your job will finish running prior to DED time, then your job can then be considered for running. To change the wall-clock time request for your job, follow the example below:

```
%qalter -l walltime=hh:mm:ss jobid
```

To find out whether the system is in dedicated time, run the command `schedule all`. For example:

```
pfe25$ schedule all
```

```
No scheduled downtime for the specified period.
```

Scheduling is Turned Off

Sometimes job scheduling is turned off by NAS Control Room staff or a system administrator. This is usually done when there are system or PBS issues that need to be resolved before jobs can be scheduled to run. When this happens, you should see the following message near the beginning of the `qstat -au your_userid` output.

```
+++Scheduling turned off.
```

Your Job Has Been Placed On Hold ("H" Mode)

A job can be placed on hold either by the job owner or by someone who has root privilege, such as a system administrator. If your job has been placed on hold by a system administrator, you should get an email explaining the reason for the hold.

Your Home Filesystem or Default /nobackup Filesystem is Down

When a PBS job starts, the PBS prologue checks to determine whether your home filesystem and default /nobackup are available before executing the commands in your script. If your default /nobackup filesystem is down, PBS cannot run your job and will put the job back in the queue. If your PBS job does not need any file in that filesystem, you can tell PBS that your job will not use the default /nobackup to allow your job to run.

For example, if your default is /nobackupp1 (for Pleiades), you can add the following in your PBS script:

```
#PBS -l /nobackupp1=0
```

Using `pdsh_gdb` for Debugging Pleiades PBS Jobs

A script called `pdsh_gdb`, created by NAS staff Steve Heistand, is available on Pleiades under `/u/scicon/tools/bin` for debugging PBS jobs *while the job is running*.

Launching this script from a Pleiades front-end node allows one to connect to each compute node of a PBS job and create a stack trace of each process. The aggregated stack trace from each process will be written to a user specified directory (by default, it is written to `~/tmp`).

Here is an example of how to use this script:

```
pfel% mkdir tmp
pfel% /u/scicon/tools/bin/pdsh_gdb -j jobid -d tmp -s -u nas_username
```

More usage information can be found by launching `pdsh_gdb` without any option:

```
pfel% /u/scicon/tools/bin/pdsh_gdb
```

Effective Use of PBS

Streamlining PBS Job File Transfers from Pleiades to Lou

Some users prefer to streamline the storage of files (created during a job run) to Lou, within a PBS job. Since Pleiades compute nodes do not have network access to the Lou storage nodes, or other nodes outside of Pleiades, all file transfers to Lou within a PBS job must first go through one of the front-ends (pfe[20-27], or bridge[1-4]).

Here is an example of what you can add to your PBS script to accomplish this:

1. **ssh** to a bridge node (for example, bridge2) and create a directory on lou[1,2] where the files are to be copied.

```
ssh -q bridge2 "ssh -q lou mkdir -p $SAVDIR"
```

Here, \$SAVDIR is assumed to have been defined earlier in the PBS script. Note the use of **-q** for quiet-mode, and double quotes so that shell variables are expanded prior to the **ssh** command being issued.

2. Use **scp** via a bridge node to transfer the files.

```
ssh -q bridge2 "scp -q $RUNDIR/* lou:$SAVDIR"
```

Here, \$RUNDIR is assumed to have been defined earlier in the PBS script.

Avoiding Job Failure from Overfilling /PBS/spool

When your PBS job is running, its error and output files are kept in the /PBS/spool directory of the first node of your job. However, the space under /PBS/spool is limited, and when it fills up, any job that tries to write to /PBS/spool may die. This makes the node unusable by jobs until the spool directory is cleaned up manually.

To avoid this situation, PBS may start enforcing a 100-MB limit on the combined sizes of error and output files produced by a job. If this policy goes into effect and a job exceeds that limit, PBS will kill the job.

To prevent this from happening to your job, do *not* write large amounts of content in the PBS output/error files. If your executable normally writes a lot of messages to either standard out or standard error, you should redirect them in your PBS script. Below are a few options to consider:

1. Redirect standard out and standard error to a single file:

```
(for csh)
mpiexec a.out >& output
(for bash)
mpiexec a.out > output 2>&1
```

2. Redirect standard out and standard error to separate files:

```
(for csh)
(mpiexec a.out > output) > error
(for bash)
mpiexec a.out > output 2> error
```

3. Redirect only standard out to a file:

```
(for both csh and bash)
mpiexec a.out > output
```

The files "output" and "error" are created under your own directory and you can view the contents of these files while your job is still running.

If you are concerned that these two files could get clobbered in a second run of the script, you can create unique filenames for each run. For example, you can add the PBS JOBID to "output" using the following:

```
(for csh)
mpiexec a.out >& output.$PBS_JOBID
(for bash)
mpiexec a.out > output.$PBS_JOBID 2>&1
```

where \$PBS_JOBID contains a number (jobid) and the name of the PBS server, such as

12345.pbspl1.nas.nasa.gov.

If you just want to include the numeric part of the PBS JOBID, do the following:

```
(for csh)
set jobid=`echo $PBS_JOBID | awk -F . '{print $1}'`
mpiexec a.out >& output.$jobid
(for bash)
export jobid=`echo $PBS_JOBID | awk -F . '{print $1}'`
mpiexec a.out > output.$jobid 2>&1
```

In the event that you do not redirect your executable's standard out and error to a file, you can see the contents of your PBS output/error files before your job completes by following the two steps below:

1. Find out the first node of your PBS job using "-W o=+rank0" for qstat:

```
%qstat -u your_username -W o=+rank0
JobID          User      Queue   Jobname   TSK Nds    wallt S    wallt  Eff Rank0
-----
868819.pbspl1 zsmith  long    ABC        512  64 5d+00:00 R 3d+08:39 100% r162i0n14
```

This shows that the first node is r162i0n14.

2. Log in to the first node and *cd* to /PBS/spool to find your PBS stderr/out file(s). You can view the contents of these files using *vi* or *view*.

```
%ssh r162i0n14
%cd /PBS/spool
%ls -lrt
-rw----- 1 zsmith a0800 49224236 Aug  2 19:33 868819.pbspl1.nas.nasa.gov.OU
-rw----- 1 zsmith a0800 1234236 Aug  2 19:33 868819.pbspl1.nas.nasa.gov.ER
```

Running Multiple Serial Jobs to Reduce Wall-Time

On Pleiades, running multiple serial jobs within a single batch job can be accomplished with following example PBS scripts. The maximum number of processes you can run on a single node will be limited to the core-count-per-node or the maximum number that will fit in a given node's memory, whichever is smaller.

Processor Types	Cores/node	Available Memory/node
Harpertown	8	7.6 GB
Nehalem-EP	8	22.5 GB
Westmere	12	22.5 GB
Sandy Bridge	16	~31.0 GB

The examples below allow you to spawn serial jobs across nodes using the **mpiexec** command. Note that a special version of **mpiexec** from the `mpi-mvapich2/1.4.1/intel` module is needed in order for this to work. This **mpiexec** keeps track of `$PBS_NODEFILE` and places each serial job onto the CPUs listed in `$PBS_NODEFILE` properly. The use of the arguments `-comm none` for this version of **mpiexec** is essential for serial codes or scripts. In addition, to launch multiple copies of the serial job at once, the use of the **mpiexec**-supplied `$MPIEXEC_RANK` environment variable is needed to distinguish different input/output files for each serial job. This is demonstrated with the use of a wrapper script **wrapper.csh** in which the input/output identifier (that is, `{rank}`) is calculated from the sum of `$MPIEXEC_RANK` and an argument provided as input by the user.

Example 1

This first example runs 64 copies of a serial job, assuming that 4 copies will fit in the available memory on one node and 16 nodes are used.

serial1.pbs

```
#PBS -S /bin/csh
#PBS -j oe
#PBS -l select=16:ncpus=4
#PBS -l walltime=4:00:00

module load mpi-mvapich2/1.4.1/intel

cd $PBS_O_WORKDIR

mpiexec -comm none -np 64 wrapper.csh 0
```

wrapper.csh

```
#!/bin/csh -f
@ rank = $1 + $MPIEXEC_RANK
./a.out < input_${rank}.dat > output_${rank}.out
```

This example assumes that input files are named *input_0.dat*, *input_1.dat*, ... and that they are all located in the directory where the PBS script is submitted from (that is, `$PBS_O_WORKDIR`). If the input files are in different directories, then `wrapper.csh` can be modified appropriately to `cd` into different directories as long as the directory names are differentiated by a single number that can be obtained from `$MPIEXEC_RANK` (=0, 1, 2, 3, ...). In addition, be sure that `wrapper.csh` is executable by you, and you have the current directory included in your path.

Example 2

A second example provides the flexibility where the total number of serial jobs may not be the same as the total number of processors requested in a PBS job. Thus, the serial jobs are divided into a few batches and the batches are processed sequentially. Again, the wrapper script is used where multiple versions of the program `a.out` in a batch are run in parallel.

serial2.pbs

```
#PBS -S /bin/csh
#PBS -j oe
#PBS -l select=10:ncpus=3
#PBS -l walltime=4:00:00

module load mpi-mvapich2/1.4.1/intel

cd $PBS_O_WORKDIR

# This will start up 30 serial jobs 3 per node at a time.
# There are 64 jobs to be run total, only 30 at a time.

# The number to run in total defaults here to 64 or the value
# of PROCESS_COUNT that is passed in via the qsub line like:
# qsub -v PROCESS_COUNT=48 serial2.pbs
#

# The total number to run at once is automatically determined
# at runtime by the number of CPUs available.
# qsub -v PROCESS_COUNT=48 -l select=4:ncpus=3 serial2.pbs
# would make this 12 per pass not 30. No changes to script needed.

if ( $?PROCESS_COUNT ) then
  set total_runs=$PROCESS_COUNT
else
  set total_runs=64
endif
```

```
set batch_count=`wc -l < $PBS_NODEFILE`

set count=0

while ($count < $total_runs)
  @ rank_base = $count
  @ count += $batch_count
  @ remain = $total_runs - $count
  if ($remain < 0) then
    @ run_count = $total_runs % $batch_count
  else
    @ run_count = $batch_count
  endif
  mpiexec -comm none -np $run_count wrapper.csh $rank_base
end
```

Checking the Time Remaining in a PBS Job from a Fortran Code

During job execution, sometimes it is useful to find out the amount of time remaining for your PBS job. This allows you to decide if you want to gracefully dump restart files and exit before PBS kills the job.

If you have an MPI code, you can call `MPI_WTIME` and see if the elapsed walltime has exceeded some threshold to decide if the code should go into the shutdown phase.

For example:

```
include "mpif.h"

real (kind=8) :: begin_time, end_time

begin_time=MPI_WTIME()
do work
end_time = MPI_WTIME()

if (end_time - begin_time > XXXXX) then
  go to shutdown
endif
```

In addition, the following library has been made available on Pleiades for the same purpose:

`/u/scicon/tools/lib/pbs_time_left.a`

To use this library in your Fortran code, you need to:

1. Modify your Fortran code to define an external subroutine and an integer*8 variable
`external pbs_time_left`
`integer*8 seconds_left`
2. Call the subroutine in the relevant code segment where you want the check to be performed
`call pbs_time_left(seconds_left)`
`print*, "Seconds remaining in PBS job:", seconds_left`

Note: The return value from **`pbs_time_left`** is only accurate to within a minute or two.

3. Compile your modified code and link with the above library using, for example:
`LDFLAGS=/u/scicon/tools/lib/pbs_time_left.a`

Using GNU Parallel to Package Multiple Jobs in a Single PBS Job

GNU is a complete, free software system, upward-compatible with Unix. GNU parallel is a shell tool for executing jobs in parallel. It uses the lines of its standard input to modify shell commands, which are then run in parallel. Detailed information about this tool can be found on the [GNU Operating System website](#) and its related [parallel man page](#).

On Pleiades, a copy of GNU parallel is available under `/usr/bin`.

The three examples below demonstrate how you can use GNU parallel to run multiple tasks in a single PBS batch job.

Example 1

This example script runs 64 copies of a serial executable file, and assumes that 4 copies will fit in the available memory of one node and that 16 nodes are used.

gnu_serial1.pbs

```
#PBS -lselect=16:ncpus=4
#PBS -lwalltime=4:00:00

cd $PBS_O_WORKDIR

seq 64 | parallel -j 4 -u --sshloginfile $PBS_NODEFILE \
  "cd $PWD;./myscript.csh {}"
```

In the above PBS script, the last command uses the parallel command to simultaneously run 64 copies of `myscript.csh` located under `$PBS_O_WORKDIR`. Here is the specific breakdown:

- seq 64

Generates a set of integers *1, 2, 3, ..., 64* that will be passed to the parallel command.

- -j 4

GNU parallel will determine the number of processor cores on the remote computers and run the number of tasks as specified by `-j`. In this case, `-j 4` tells the parallel command to run 4 tasks in parallel on one compute node.

- -u

Tells the parallel command to print output as soon as possible. This may cause output from different commands to be mixed. GNU parallel runs faster with `-u`. This can be reversed with `--group`.

- `--sshloginfile $PBS_NODEFILE`

Distributes tasks to the compute nodes listed in `$PBS_NODEFILE`.

- `"cd $PWD; ./myscript.csh {}"`

Changes directory to the current working directory and runs `myscript.csh` located under `$PWD`. At this point, `$PWD` is the same as `$PBS_O_WORKDIR`. The `{}` is an input to `myscript.csh` (see below) and will be replaced by the sequence number generated from `seq 64`.

myscript.csh

```
#!/bin/csh -fe
date
mkdir -p run_$1
cd run_$1

echo "Executing run $1 on" `hostname` "in $PWD"

$HOME/bin/a.out < ../input_$1 > output_$1
```

In this above sample script executed by the parallel command:

- `$1` refers to the sequence numbers (1, 2, 3, ..., 64) from the `seq` command that was piped into the parallel command
- For each serial run, a subdirectory named `run_$1` (`run_1`, `run_2`, ...) is created
- The echo line prints information back to the PBS `stdout` file
- The serial `a.out` is located under `$HOME/bin`
- The input for each run, `input_$1` (`input_1`, `input_2`, ...) is located under `$PBS_O_WORKDIR`, which is the directory above `run_$1`
- The output for each run (`output_1`, `output_2`, ...) is created under `run_$1`

Potential Modifications to Example 1

There are multiple ways to pass arguments to parallel. For example, instead of using the `seq` command to pass a sequence of integers, you can also pass in a list of directory names or filenames using `ls -1` or `cat mylist`, where the file `mylist` contains a list of entries.

Example 2

This script is similar to Example 1, except that 6 nodes are used instead of 16 nodes. This means that 24 serial `a.out`s can be run simultaneously, since a total of 24 cores (6 nodes x 4 cores) are requested. As each `a.out` completes its work on a core, another `a.out` is launched by the parallel command to run on the same core.

`mymyscript.csh` is the same as that shown in the previous example.

gnu_serial2.pbs

```
#PBS -lselect=6:ncpus=4
#PBS -lwalltime=4:00:00

cd $PBS_O_WORKDIR

seq 64 | parallel -j 4 -u --sshloginfile $PBS_NODEFILE \
  "cd $PWD; ./mymyscript.csh {}"
```

Example 3

In this example, an OpenMP executable is run with 12 OpenMP threads on one Westmere node. To run 64 copies of this executable with 10 copies running simultaneously on 10 nodes:

gnu_openmp.pbs

```
#PBS -lselect=10:ncpus=12:mpiprocs=1:ompthreads=12:model=wes
#PBS -lwalltime=4:00:00

cd $PBS_O_WORKDIR

seq 64 | parallel -j 1 -u --sshloginfile $PBS_NODEFILE \
  "cd $PWD; ./myopenmpscript.csh {}"
```

myopenmpscript.csh

```
#!/bin/csh -fe
date
mkdir -p run_$1
cd run_$1

setenv OMP_NUM_THREADS 12

echo "Executing run $1 on" `hostname` "in $PWD"

$HOME/bin/a.out < ../input_$1 > output_$1
```