To help you prepare for running jobs on Electra's Skylake compute nodes, this short user guide includes information on the general configuration of Skylake nodes, compiling your code, running PBS jobs, and checking allocation usage.

Overview of Electra Skylake Nodes

Electra includes 2,196 Skylake nodes, which are partitioned into sixteen physical racks. Each node contains two 20-core Xeon Gold 6148 sockets (2.4 GHz) and 192 GB of memory. The nodes are connected to the Electra InfiniBand network (ib0 and ib1) via four-lane Enhanced Data Rate (4X EDR) devices and switches for internode communication. The ib1 fabric is used mainly for I/O and is connected to the NAS Lustre filesystems. In addition, Electra has access to the NAS compute resources home filesystems, front-end systems (PFEs), and PBS server (pbspl1). You can access Electra only through PBS jobs submitted from the PFEs.

Compiling Your Code For Skylake Nodes

The Skylake processors include the Advanced Vector Extensions 512 (AVX-512). Intel AVX-512 optimizations are included in Intel compiler version 16.0 and later versions. We recommend that you test the latest Intel compiler, which may include better optimizations for AVX-512.

For Skylake-specific optimizations, use the compiler option -xCORE-AVX512.

If you want a single executable that will run on any of the Electra processor types, with suitable optimization to be determined at runtime, you can compile your application using the option:
-O3 -axCORE-AVX512,CORE-AVX2 -xAVX.

Note: The use of either of these options, -xCORE-AVX512 or -axCORE-AVX512, could either improve or degrade performance of your code. Be sure to check performance with and without these flags before using them for production runs.

Running PBS Jobs on Electra Skylake Nodes

To request Electra Skylake nodes, use :model=sky_ele in your PBS script, as shown in the example below.

Important: Because MPT 2.15 and earlier versions do not support the ConnectX-5 host channel adapters (HCAs), the environment variables MPI_IB_XRC and MPI_XPMEM_ENABLED have been disabled for jobs running on Skylake. If your MPI applications perform significant MPI collective operations and rely on having these two variables enabled to get good performance, use MPT 2.16 or newer versions. To use the NAS-recommended MPT version, use the command:
module load mpi-hpe/mpt

Sample PBS Script For Electra Skylake Nodes

```
#PBS -l select=10:ncpus=40:mpiprocs=40:model=sky_ele
#PBS -l walltime=8:00:00
#PBS -q normal

module load mpi-hpe/mpt
module load comp-intel/

cd $PBS_O_WORKDIR

mpiexec -np 400 ./a.out
```

Checking Allocation Usage For Electra Skylake Jobs

To track allocation usage for jobs running on Electra Skylake nodes, run:

```
acct_query -c electra_S
```

Examples:

- To track the total SBUs usage for one of your GIDs (for example, s0001) since 9/30/17:

  ```
  % acct_query -c electra_S -ps0001 -b 9/30/17
  ```

- To track the SBUs usage for each of your jobs run today:

  ```
  % acct_query -c electra_S -olow
  ```

For more information, see Skylake Processors.

To help you prepare for running jobs on Electra's Broadwell compute nodes, this short review includes information on the general configuration, compiling your code, running PBS jobs, and checking allocation usage.

Overview of Electra Broadwell Nodes

Electra is housed in a module outside of the primary NAS facility. The system has 16 Broadwell racks, each containing 72 nodes. Each of the 72 nodes contains two 14-core E5-2680v4 (2.4 GHz) processor chips and 128 GB of memory.

Electra's InfiniBand fabric (ib1) is used mainly for I/O and is connected to the NAS Lustre filesystems. In addition, Electra and all other NAS compute systems share the same home filesystems. It also share the front-end systems (PFE) and PBS server (pbspl1) with some NAS compute nodes such as sky_ele, sky_el, and rom_ait. You can access Electra only through PBS jobs submitted from the PFEs.

Usage of Electra Broadwell nodes is charged to your project's HECC Standard Billing Unit (SBU) allocation.

Compiling Your Code For Broadwell Nodes

Like the Haswell processors, the Broadwell processors support the Advanced Vector Extensions 2 (AVX2) instructions, in addition to AVX (introduced with Sandy Bridge processors), SSE4.2 (introduced with Nehalem processors), and earlier generations of SSE.

AVX2 supports floating point fused multiply-add, integer vector instructions extended to 256-bit, and vectorization gather support, among other features. Your application may be able to take advantage of AVX2, however, not all applications can make effective use of this set of instructions. We recommend that you use the latest Intel compiler, by using the command module load comp-intel and experiment with the following sets of compiler options before making production runs on the Broadwell nodes:

- -O*2* -xCORE-AVX2
- -O*3* -xCORE-AVX2

These compiler options generate an executable that is optimized for running on Broadwell and Haswell, but cannot run on any of the older processor types. If you prefer to generate a single executable that will run on many Intel processor types (Sandy Bridge, Ivy Bridge, Haswell, Broadwell, Skylake, and Cascade Lake), try using one of the following sets of compiler options:

- -O*2* (or -O*3*)
- -O*2* (or -O*3*) -axCORE-AVX512,CORE-AVX2 -xAVX

The use of -axCORE-AVX512,CORE-AVX2 allows the compiler to generate multiple code paths with suitable optimization to be determined at run time.

If your application does not show performance improvements with -xCORE-AVX2 or -axCORE-AVX512,CORE-AVX2 -xAVX (as compared with just -O2 or -O3) when running on Broadwell nodes, it is more advantageous to use the executable built with just -o2 or -o3 for production runs.

TIP: You can add the compiler options -ip or -ipo to instruct the compiler to look for ways to better optimize and/or vectorize your code. Also, to generate a report on how well your code is vectorized, add the compiler flag -vec-report2.

Notes:

- If you have an MPI code, we strongly recommend that you load the mpi-hpe/mpt module, which always points to the NAS recommended MPT version.
- Ensure your jobs run correctly on Broadwell nodes before you start production work.

Running PBS Jobs on Broadwell Nodes

To request Broadwell nodes, use model=bro_ele in your PBS script.

```
#PBS -l select=10:ncpus=28:mpiprocs=28:model=bro_ele
#PBS -l site=static_broadwell
```

Sample PBS Script For Broadwell Nodes

```
#PBS -l select=10:ncpus=28:mpiprocs=28:model=bro_ele
#PBS -l walltime=8:00:00
#PBS -q normal
module load comp-intel/2020.4.304 mpi-hpe/mpt
cd $PBS_O_WORKDIR
mpiexec -np 280 ./a.out
```

For more information about Broadwell nodes, see:

- Electra Configuration Details
- Broadwell Processors

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

For example, suppose your typical job requires 5 days of wall time. If there are fewer than 5 days before the start of dedicated time, the job won'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. When no such time slot is found between now and the dedicated time, it will look for shorter and shorter time slots, down to the minimum wall time 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 job_id
```

You can also use the `qalter` command to change the wall time:

```
qalter -l walltime=hh:mm:ss job_id
```

Note: For jobs on Endeavour, be sure to include both the job sequence number and the PBS server name, pbspl4, in the *job_id* (for example, 2468.pbspl4).

If you have any questions or problems, please contact the NAS Control Room at (800) 331-8737, (650) 604-4444, or by email at support@nas.nasa.gov.

During job execution, it can be 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 wall time has exceeded some threshold in order 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 NAS compute systems 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
   ```

Many jobs running across multiple nodes perform setup and pre-processing tasks on the first node, then launch computations across many nodes, and finish with post-processing and cleanup steps running only on the first node. The post-processing and cleanup steps may take minutes or even hours, during which the remaining nodes in the job are idle. You can use the pbs_release_nodes command to release these idle nodes so that you are not charged for them, and they can be made available to other jobs.

Some users have already taken steps to minimize the number of idle nodes by separating their workflow into multiple jobs of appropriate sizes. The downside to this approach is additional time waiting in the queue for these multiple jobs. Using pbs_release_nodes may allow these jobs to be combined, reducing the amount of wait time in the queue.

Node Tracking and Job Accounting

When you use the pbs_release_nodes command, the released nodes are removed from the contents of $PBS_NODEFILE. The job accounting system tracks when the nodes are released, and adjusts the amount of SBUs charged accordingly.

Running the pbs_release_nodes Command

You can either include the pbs_release_nodes command in your job script or you can run it interactively, for example, on a PFE. If you run it in your job script, you can use the $PBS_JOBID environment variable to supply the *job_id*. You can use pbs_release_nodes multiple times in a single job.

There are two ways to use the pbs_release_nodes command. In both cases, the first node is retained.

- **Keep the first node and release all other nodes**
  If you want to retain the first node only, run the command as follows to release the remaining nodes:

  pfe21% pbs_release_nodes -j *job_id* -a

- **Release one or more specific nodes**
  If you want to retain one or more nodes in addition to the first node, run the command as follows to release specific idle nodes:

  pfe21% pbs_release_nodes -j *job_id node1 node2 ... nodeN*

To find out which nodes are being used by a job before you run the pbs_release_nodes command, use one of the following methods:

- **For interactive jobs:** Run qstat -f *job_id*. In the output, look for the nodes listed under the exec_vnode job attribute.
- **For job scripts:** Run cat $PBS_NODEFILE. The output provides a list of the nodes in use.

Note: $PBS_NODEFILE may have duplicate entries for nodes, depending on how resources were requested. See **man pbs_resources** for details.

Sample Job Script with pbs_release_nodes Command

The following sample job script keeps the first node and releases all other nodes before running post-processing steps:

```
#PBS -S /bin/csh
#PBS -N cfd
#PBS -l select=4:ncpus=8:mpiprocs=8:model=sky_ele
#PBS -l walltime=4:00:00
#PBS -j oe
#PBS -W group_list=a0801
#PBS -m e
module load comp-intel/
module load mpi-hpe/mpt
cd $PBS_O_WORKDIR
mpiexec dplace -s1 -c 16-23 ./grinder < run_input > intermediate_output
# Before executing some post-processing steps that will run only on the
# first node of the job you can release all other nodes
pbs_release_nodes -j $PBS_JOBID -a
# Now run post-processing steps
./post_process < intermediate_output > final_output
# -end of script-
```

When you request PBS resources for your job, you must specify one or more Aitken or Electra node model (processor) types, so it is helpful to keep this information in mind:

- Usage charges on each HECC node type are based on a common Standard Billing Unit (SBU).
- The total physical memory of each node type varies from 128 GB to 512 GB.

The following table provides the SBU rate and memory for each Aitken and Electra node type:

Processor Type	SBU Rate/Node	Memory/Node
Milan	4.38 (128 cores/node)	512 GB
Rome	4.06 (128 cores/node)	512 GB
Cascade Lake	1.64 (40 cores/node)	192 GB
Skylake	1.59 (40 cores/node)	192 GB
Broadwell	1.00 (28 cores/node)	128 GB

Note: The amount of memory available to a PBS job is less than the node's total physical memory, because the system kernel can use up to 4 GB of memory in each node.

Checking Available Nodes Before Requesting Resources

If your application can run on any of the Aitken or Electra processor types, you might be able to reduce wait time by requesting a type that has more free nodes (nodes currently unoccupied by other running jobs).

Before you submit your job, you can find out the current numbers of used and free nodes for each processor type by running the node_stats.sh script, which is accessible by loading the scicon/cli_tools modulefile. For example:

```
% module load scicon/cli_tools
% node_stats.sh
```

You can also identify which processor models are in use for each job currently running on HECC systems. Run the following command, and check the Model field in the output:

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

Resource Request Examples

As shown in these examples, use the model=[bro_ele,sky_ele,cas_ait,rom_ait, mil_ait] attribute to request the node model type(s) for your job.

The default operating system for all model types is TOSS (Tri-Lab Operating System Stack). For all model types, the default is aoe=toss4:aoe=toss4 to your PBS resource request.)

Example 1: Basic Resource Request

Each of the following sample command lines requests a single node model type for a 128-process job:

```
#PBS -l select=5:ncpus=28:model=bro_ele
#to run all 28 cores on each of the 5 Broadwell nodes
#(12 cores in the 5th node will go unused)

#PBS -l select=4:ncpus=40:model=sky_ele
#to run all 40 cores on each of the 4 Skylake nodes
#(32 cores in the 4th node will go unused)

#PBS -l select=4:ncpus=40:model=cas_ait
#to run all 40 cores on each of the 4 Cascade Lake nodes
#(32 cores in the 4th node will go unused)

#PBS -l select=1:ncpus=128:model=rom_ait
#to run all 128 cores on 1 Rome node

#PBS -l select=1:ncpus=128:model=mil_ait
#to run all 128 cores on 1 Milan node
```

You can specify both the queue type -q normal, debug, long, low, or devel and the processor type. For example:

```
#PBS -q normal
```

```
#PBS -l select=128:ncpus=128:model=mil_ait
```

Example 2: Requesting Different ncpus in a Multi-Node Job

For a multi-node PBS job, the number of CPUs `ncpus` used in each node can be different. This feature is useful for jobs that need more memory for some processes, but less for other processes. You can request resources in "chunks" for a job with varying `ncpus` per node.

This example shows a request of two resource chunks. In the first chunk, one CPU in one Broadwell node is requested, and in the second chunk, 28 CPUs in each of three other Broadwell nodes are requested:

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

Example 3: Requesting Multiple Node Model Types

A PBS job can run across more than one node model type if the nodes are all from Electra or all from Aitken, but not across the systems. 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

You can accomplish this by specifying the resources in chunks, with one chunk requesting one node model type and another chunk requesting a different type.

This example shows a request for one Skylake node (which provides ~185 GB of memory per node) and three Broadwell nodes (which provide ~120 GB of memory per node):

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

#PBS -lselect=1:ncpus=24:mpiprocs=24:model=sky_ele+3:ncpus=28:mpiprocs=28:model=bro_ele
```

TEMPORARY NOTICE: The sky_wide queue on Electra's Skylake nodes has been disabled until further notice in order to process high-priority wide jobs.

The sky_wide queue is intended to help you run very wide ("at scale") jobs on Electra's Skylake nodes by ensuring that at least once a week, the jobs in the queue will have enough available nodes to run. Job requests submitted to the sky_wide queue must specify between 576 and 1,152 Skylake nodes (between 25% and 50% of the total number of Skylake nodes, which is currently 2,304). The maximum wall time is currently 5 days.

Submitting Your Job to the sky_wide Queue

You can submit jobs to the queue each week. Jobs submitted to the queue by Thursday at 5:00 p.m. (Pacific Time) are scheduled to run starting the following Monday at 10:00 a.m. PBS runs the jobs one at a time, prioritized by job size; wider jobs are run before those requesting fewer nodes.

When it is your job's turn to run, a reservation is created for the number of nodes requested, plus a few extra nodes. The job is then moved from the sky_wide queue into the reservation's queue, where it starts running.

If the job fails early, you can submit a replacement job to the reservation queue without losing your place in line. Eventually, the queue will go idle or the reservation will expire, and the next job will be selected.

Important: Jobs submitted to the sky_wide queue after Thursday at 5:00 p.m. (Pacific Time) will not be scheduled to run the following Monday; rather, they will carry over to the next week.

Sample Job Submission Script

The following sample script includes several instructions. Each line is described below the script.

```
#PBS -q sky_wide
#PBS -l select=NNN:model=sky_ele:ncpus=nn:aoe=sles12
#PBS -l max_walltime="24:00:00",min_walltime="5:30:00"
#PBS -m abe
#PBS -l site="extra_nodes=4:extra_time=20:idle_timeout=20"
```

Job Submission

The first two lines request the sky_wide queue and submit your job:

```
#PBS -q sky_wide
#PBS -l select=NNN:model=sky_ele:ncpus=nn:aoe=sles12
```

Wall Time Request

This line specifies how much wall time the job needs. Keep in mind that if the job fails, you can submit a replacement job, but its wall time must fit within the time remaining from the original job. By specifying a maximum and minimum useful wall time in the original job script, you can use the same script, unedited, for the replacement job. The sample script requests a total of 24 hours of time, and any replacement job will not run unless at least 5.5 hours of wall time remain:

```
#PBS -l max_walltime="24:00:00",min_walltime="5:30:00"
```

Note: This technique, a PBS feature known as shrink-to-fit, is described in the article Running Jobs Before Dedicated Time.

Notification

If there are several jobs in the sky_wide queue, your job might not be the first to run. You can tell PBS to send you an email notification when your job starts, ends, or aborts, so you'll know whether you need to submit a replacement job:

```
#PBS -m abe
```

Additional Options

PBS automatically adds a few extra nodes to the job reservation, beyond the number of nodes in the request, in case nodes fail during the run. That way, a replacement job will still have enough reserved nodes. The default is 12 nodes. To specify a different value, use site=extra_nodes=nn:

```
#PBS -l site=extra_nodes=4
```

Recommendation: There are four Skylake nodes per blade. Because it is possible for a failing node to take down the whole blade, it makes sense to set aside extra nodes in multiples of four.

PBS also adds extra time to each job reservation. The default is 15 minutes. To specify different value, use site=extra_time=nn:

```
#PBS -l site=extra_time=20
```

Once your per-job reservation starts, PBS monitors the reservation. If the reservation queue is idle for a set number of minutes, the reservation is terminated early. The default is 30 minutes. To specify a different value, use site=idle_timeout=nn:

```
#PBS -l site=idle_timeout=20
```

To specify multiple site options in one line, separate them by colons (":"), as shown in the sample script:

```
#PBS -l site="extra_nodes=4:extra_time=20:idle_timeout=20"
```

Note: When you consider these requests, keep in mind that your account is billed for the full number of nodes and for the actual duration of the job reservation.

Checking Status of Wide Jobs

Run the pbs_scale_jobs command, which uses qstat to display the status of jobs submitted to the sky_wide queue. For example:

```
$ pbs_scale_jobs
                                        Req'd    Elap
JobID       User     Queue    Jobname    TSK Nds wallt S wallt Eff
----------- -------- -------- --------- ----- --- ----- - ----- ---
3827.testpbs zsmith   R3853    testing3 24000 600 10:15 Q 02:26  --
3792.testpbs zsmith   sky_wide testing1 23040 576 00:15 Q 03:11  --
3808.testpbs zsmith   sky_wide testing2 23040 576 00:15 Q 03:11  --

== Jobs to consider next cycle ==
                                        Req'd    Elap
JobID       User     Queue    Jobname    TSK Nds wallt S wallt Eff
----------- -------- -------- --------- ----- --- ----- - ----- ---
4001.testpbs zsmith   sky_wide testing5 23040 576 00:15 Q 01:10  --
4002.testpbs zsmith   sky_wide testing6 23040 576 00:15 Q 01:10  --
```

In this example, the jobs named testing1-3 were submitted before the cutoff time. Jobs testing5 and testing6 were submitted too late for this week's run and will be considered next week.

The testing3 job has been moved to its run queue (R3853, in this example). Note that although testing3 was submitted after testing1 and 2, the PBS scheduler runs it first because it uses more nodes than the other jobs.

Submitting Replacement Jobs

Replacement jobs are submitted to the reservation queue associated with the original job, rather than to the sky_wide queue. This queue will have a name like R12345, which changes for each reservation.

However, there is an easier way to submit a replacement job to the reservation queue. You can use the queue's alias, which takes the form NAS_job_*username* (where *username* is your NAS user name).

Also, instead of editing your qsub script to change the queue name when you submit the replacement, you can use the original job script, but override the queue on the command line. For example:

```
qsub -q NAS_job_zsmith job_script
```

Running Jobs on Athena

To run a PBS job on the Athena compute nodes, submit the job from one of the Athena front-end systems (ATHFEs) to one of the queues managed by the PBS server pbs06a: devel, normal, long, and wide.

Note: You cannot submit jobs to run on Athena from a PFE.

Running Jobs on Aitken and Electra

To run a PBS job on the Aitken or Electra compute nodes, submit the job from one of the PFEs to one of the queues managed by the PBS server pbspl1.

Aitken Milan Nodes: You can submit batch jobs to run on Aitken's Milan nodes (mil_ait) either from a PFE or from one of the two Aitken Milan front-end systems (afe01, afe02). Interactive PBS jobs for Aitken Milan nodes must be submitted from the AFEs because currently there is no direct network connection to the PFEs.

Running Jobs on Endeavour

To run a PBS job on the shared memory systems Endeavour3/4, submit the job from a PFE to one of the queues managed by the PBS server pbspl4.

Running Jobs on Cabeus

To run a PBS job on:

- Milan A100 (mil_a100) submit the job from a Cabeus front end (cfe01 or cfe02) to one of the queues managed by the PBS server pbspl4.
- Grace Hopper nodes (gh200), submit your job from a Cabeus Grace Hopper front end (cghfe01 or cghfe02) to one of the queues served by the PBS server pbs05a.

PBS Queues

To view a complete list of all queues, including their defaults or limits on the number of cores, wall time, and priority, run theqstat command from an appropriate front-end system as follows:

```
qstat -Q
```

To view more detailed information about each queue, such as access control, defaults, limits on various resources, and status:

```
qstat -fQ queue_name
```

Brief summaries of the various queues are provided in the following sections.

Queues for Athena Users

You can run jobs on Athena using any of the following queues:

```
Queue   NCPUs/      Time/
name     max/def    max/def    pr
======= =====/=== ======/====== ===
devel    256/ 1    02:00/--     100
normal    --/ 1    08:00/--       0
long      --/ 1   120:00/--       0
wide     128/64    24:00/--      20
```

Queues for Aitken and Electra Users

You can run jobs on Aitken and Electra using any of the following queues:

```
Queue   NCPUs/      Time/
name     max/def    max/def    pr
======= =====/=== ======/====== ===
low       --/  8  04:00/ 00:30 -10
normal    --/  8  08:00/ 01:00   0
long      --/  8 120:00/ 01:00   0
debug     --/  8  02:00/ 00:30  15
devel     --/  1  02:00/    -- 101
```

The normal, long and low queues are for production work. The debug and devel queues have higher priority and are for debugging and development work.

For the debug queue, each user is allowed a maximum of 2 running jobs using a total of up to 128 nodes. For the devel queue, each user is allowed only 1 job (either queued or running).

Queues for Cabeus GPU Users

For information about queues for the Cabeus GPU nodes, see Requesting GPU Resources.

See also:

- Preparing to Run on Cabeus Grace Hopper Nodes
- Preparing to Run on Cabeus A100 GPU Nodes

Queues for Endeavour Users

Jobs submitted from PFEs to the e_normal, e_long, e_debug, and e_low queues are routed to run on the Endeavour shared-memory system, which has its own PBS server.

See Endeavour PBS Server and Queue Structure for more information.

Queue for Lou Users

The ldan queue provides access to one of the Lou data analysis nodes (LDANs) for post-processing archive data on Lou.

The ldan queue is managed by the same PBS server as all of the Electra and Aitken queues and is accessible by submitting jobs from an LFE or PFE.

Each job on the ldan queue is limited to 1 node and 72 hours. Each user can have a maximum of 2 running jobs on this queue.

See Lou Data Analysis Nodes for more information.

Queues with Restricted Access

The vlong Queue

Access to the vlong queue requires authorization from the NAS User Services manager.

A maximum of 1024 nodes and 16 days are allowed on this queue.

To avoid losing all useful work in the event of a system or application failure, ensure that your application includes checkpoint and restart capability; this is especially important when using this queue.

The wide Queue

The wide queue is a high priority queue used for jobs that meet either one of the following conditions:

- The number of requested NCPUs (number of nodes * number of cores per node) exceeds 25% of the user's mission shares
- The number of nodes requested of a processor model exceeds 25% of the total number of nodes of that model. See the Electra and Aitken configuration pages for the total number of nodes for each model.

Users do not have direct access to the wide queue. Instead, a job that meets the above criteria is moved into the wide queue by NAS staff. Only 1 job in the wide queue is allowed among Electra and Aitken. When more than one job meets the criteria, preference is given to the job that has been waiting for longer time or whose owner does not have other jobs running.

Athena: You can submit jobs to the Athena wide queue (with minimum of 64 nodes and maximum of 128 nodes for up to 24 hours) without requesting approval from NAS staff.

Various Special Queues

Special queues are higher priority queues for time critical projects to meet important deadlines. Some examples of special queues are armd_spl, astro_ops, heomd_spl, kepler, smd1, smd2, smd_ops, and so on. Request for creation and access of a special queue is carefully reviewed by the mission directorate or by the program POC who authorizes the project's SBU allocation. The request is rarely granted unless the project's principal investigator demonstrates that access to a special queue is absolutely necessary.

Advanced Reservation Queues

An advanced reservation queue allows you to reserve resources for a certain period ahead of time. One circumstance in which the NAS User Services manager may authorize the use of a reservation queue is when a user plans to run an extra-large job, such as one exceeding 15,000 cores, which would never start running without a reservation.

A reservation queue is created by NAS staff for the user and is normally named with the letter R followed by a number.

To find the status of existing PBS advanced reservations, issue the pbs_rstat command on a PFE.

Note: Queues such as alphatst, ded_time, diags, dpr, idle, resize_test, and testing are used for staff testing purposes, and are not accessible by general users.

NAS provides a special queue on Electra and Aitken, called the devel queue, that is designated for development work. Using this queue for development work provides faster turnaround than using other, non-designated queues.

Note: The devel queue should not be used for production work. Misuse of the devel queue may result in loss of access to the queue. See devel Queue Policies for more information.

Flexible Node Types

From the devel queue, you can access nodes of all Aitken and Electra processor types.

Requesting Resources

Because the devel queue does not have a default processor type, you must specify the processor type by including the :model=*xxx* attribute in your PBS resource request (where *xxx* is bro_ele, sky_ele, cas_ait or rom_ait. Note that mil_ait is not allowed to be used in devel queue.) For example, to request a Rome node:

```
#PBS -q devel
#PBS -l select=1:ncpus=128:model=rom_ait
```

Submitting Jobs

To submit your job to the devel queue, run the qsub command:

```
pfe21% qsub -q devel job_script
1234.pbspl1.nas.nasa.gov
```

To check the status of a job submitted to the devel queue, run the qstat command:

```
pfe21% qstat -u your_username devel
```

Determining Available Resources

In order to obtain the fastest possible start time, run the node_stats.sh script to view the number and types of nodes currently available in the devel queue before you submit your job. For example:

```
pfe21% node_stats.sh
```

devel Queue Policy

When using the devel queue, be aware of the following policy:

- Each user is allowed only one job on the devel queue, either running or queued. You cannot stack up a list of queued jobs or create self-submitting jobs.
- The maximum wall-clock time of a job is always 2 hours.
- A job may request no more than 512 nodes, either of a single processor/node type or a mixture of different processor/node types.
- The PBS scheduler predominately considers which users have used the most devel queue resources to decide which jobs to run next.
- devel queue usage is applied to your mission share.

The devel queue is intended for developing and testing code and cases to prepare them for production runs in other queues, such as the normal and long queues. It should not be used for performing parameter studies or for breaking a long production run into multiple consecutive runs that each fit within the 2-hour limit.

Disregarding these devel queue policies and guidelines may result in loss of access to the queue.