HPC User Documentation

PBS Queue System

WARNING: Loading certain modules (those that require a different version of openssl) can break queue system commands with an error like "qsub: symbol lookup error: /usr/lib64/libk5crypto.so (http://libk5crypto.so).3: undefined symbol: EVP_KDF_ctrl, version OPENSSL_1_1_1b". The fix is to type "module purge" then run the queue system commands. Sometimes you have to log out and log back in. If you have modules loaded in your login files (such as .bashrc.local) you may have to comment those out, then log out and log back in. If you have installed software in your account (i.e. anaconda or miniconda), you may have to comment out those lines in your login files then log out and log back in.

Nearly all supercomputing facilities use a job queue system. As users simultaneously access the supercomputer, a queue system is necessary to prevent user workloads from interfering with each other. A job queue system may also provide solutions on how to fairly allocate resources. Similar to that of a printer queue, job queue system software will start each job when the necessary resources become available. Resources may include processors, memory, and sometimes software licenses.

Tip: A queue system is a very valuable tool for users. Learning how to utilize the queue system effectively will yield efficient results when attempting to process large workloads.

Before the invention of queue systems, supercomputer users would frequently find themselves having to manually log in and monitor when resources were available. Today however with the use of a queue system, users are able to log in, submit their jobs, and allow for the queue system to monitor and apply available resources for them. The queue system will also guarantee that the user's job will get the number of CPUs and memory that they requested when the job was submitted.

Reminder: Any program running interactively (without using the queue system) on the login nodes, is limited to 10 minutes of CPU time. After 10 minutes, interactive jobs are automatically killed. Any job larger than this, or using more than 2 cores, must be run through the queue system.

Portable Batch System (PBS for short), is the queue system used by ASAX at the Alabama Supercomputer Center. Altair's PBS Professional, a commercial version of PBS, is one of the industry leading work load managers. Unlike SLURM, the previous job queue system of Alabama Supercomputer, PBS existed within the supercomputing world for several decades. Used among the top 500 supercomputers and small clusters alike, PBS is tested to scale easily from a minimal number of cores and nodes, to capacity beyond thousands of cores and nodes.

NOTE: Each queue system has its own particular set of commands used to request resources. **Commands previously used to communicate with the SLURM queue system will differ from PBS.** To ease the transitional process to ASAX, the Alabama Supercomputer Center staff provide run scripts to function with the current queue system. The run scripts offer users the convenience of not having to deal with the complex points of the queue system directly. Users who opt into using the provided run scripts will not see much of a difference in job submission. However, if you write your own queue scripts, a bit more work may be required to adapt those scripts to work the new PBS queue system.

WARNING: If you see error messages when you login via ssh, those same errors will prevent your jobs from running correctly through the queue system.

Selecting a Queue

There are a number of queues available. A list of the queues available can be displayed with the "**qlimits**" command. The qlimits command can be called without any arguments, or with a "**-a**" flag to see additional information. Calling qlimits gives an output like the following

```
Queue                     Wall Time        Mem  # Cores
------------------------- ---------- -------- --------
express                      4:00:00      16gb      1-4
small                       60:00:00       4gb      1-8
medium                     150:00:00      16gb     1-16
large                      360:00:00     120gb    1-128
bigmem                     360:00:00  130-500gb    1-32
benchmark                   24:00:00     120gb     1-20
gpu                        360:00:00     120gb     1-24
classgpu                     2:00:00       8gb      1-4
class                       12:00:00      64gb     1-60
special                   1008:00:00    2000gb     1-64
commercial                1008:00:00     360gb    1-128

Login node limits:
                           Wall Time        Mem  # Cores
```

```
-----------------------  ---------- -------- --------
Use on login nodes         00:10:00      4gb        2
```

The “Wall Time” column in this output gives the amount of time that can be requested in the format HH:MM:SS. The “Mem” column shows the maximum memory that can be requested, which is a total for all CPUs. The “# Cores” column shows how many CPU cores can be requested by a job in that queue.

NOTE: There is always an implied minimum memory of 1mb times the number of processor cores for the job.

The output you see from the qlimits command may not be identical to the example above. By default, qlimits only shows queues that are available to you. When qlimits is called with the -a flag, it shows all queues on the system, even if you don't have permissions to submit jobs to those queues.

Express Queue: Accessible to everyone on the system. Express jobs are limited to 4 hours of wall clock time. Only one job per person can be in a run state in the express queue. This makes it unusable for the majority of research calculations. However express jobs get into a run state almost immediately, making it perfect for testing purposes.

Tip: An old supercomputer user's trick is to submit a test job to the express queue, let it run a few minutes, then kill it. This is done as a check on whether the input file is constructed correctly, as incorrect inputs typically cause the calculation to fail within the first few minutes. Once this check on correct inputs is made the job can be submitted to the appropriate queue to allow it to run to completion. Unfortunately, this isn't as useful for estimating memory needs or parallel efficiency since many complex programs do non-parallel, low memory work first.

Small, Medium, and Large Queues: Are available to all users of the system. These queues are used to run the majority of the work on the supercomputers.

Tip: These queues are designed for large volumes of research work.

Class Queue: Is available for working on course homework assignments. It is typically only accessible to class accounts. These accounts only exist for the duration of the semester, and contain the letters “cls” in the account name. Jobs requiring GPU math coprocessors can also be submitted to the class queue. The class queue has access to A100 GPUs.
Note: Instructors wishing to get class accounts for their students should contact the staff at the Alabama Supercomputer Center by emailing **hpc@asc.edu** (mailto:hpc@asc.edu)

gpu Queue: Should only be used to submit jobs that require the use of a GPU math coprocessor. Two models of GPU are available, Ampere A100 GPUs and H100 GPUS.

classgpu Queue: Is available for homework assignments using GPUs. It has Ampere A100 GPUs only.

Tip: The run_gpu command gives the user the option of specifying the GPU model or either.

WARNING: The gpu queue will let a job request two GPUs. However, this only works if the software is written to use multiple GPUs. Don't request more than one GPU unless you have read the software manual, or our documentation, to verify the software is capable of using multiple GPUs.

Benchmark Queue: Gives the job exclusive use of the entire node(s) (not just the cores assigned to it). This is sometimes needed for measuring the performance of algorithms in software development. However, jobs must wait much longer to get into a run state in the benchmark queue because more jobs ahead of them must complete first.

Special Queue: Available for academic research that requires resources beyond those available through the large queues. Access to the special queue is turned on for a six month period of time, after having been granted access to this queue. In order to get access to the special queue, the user must first do a time complexity calculation to estimate the amount of CPU time and memory required by their job. They must then have their research adviser send a request for special queue access, including the resource requirements and a description of the work to **hpc@asc.edu** (mailto:hpc@asc.edu) The special queue allows only one job to run at a time per user. This is done because the system has capacity to allow a few people to be running exceptionally large jobs, but can't support allowing all users to run jobs of this magnitude.

NOTE: There is a quirk of very big memory queues like special. A 1 core job can request a maximum of 1999gb memory. Jobs with 2 or more cores can request up to the queue maximum.

A second mechanism for running exceptionally large jobs is to request dedicated machine time. Dedicated time means having the entire resources of the Alabama Supercomputer Center (or one of the clusters) reserved for the use of just one person. This is done for a once in a lifetime type of opportunity. For example, the last use of dedicated machine time was by a UAH professor who had their experiment flying on the space shuttle, and thus had to get data from the shuttle, use that data to run a simulation, and use the simulation results to call back up to the mission specialist on the shuttle to alter the experimental settings. Getting dedicated time requires months of prior planning, proposals and arrangements.

A third way of getting larger than normal resources is to do a collaborative computing hardware purchase. A researcher with computing needs beyond what the existing facilities can accommodate can work with the Alabama Supercomputer Authority to contribute money towards the purchase of additional computing resources. There would then be queues that are only accessible to the members of that research group, which has additional CPUs reserved for their usage. For example, there were once queues named dixon-serial and dixon-parallel, which were for the use of researchers working for Dr. David Dixon at the University of Alabama. One advantage of doing this is that the staff at the Alabama Supercomputer Center can take care of system administration, hardware maintenance and software installation. Another advantage is that it is possible to run work that utilizes both the resources purchased by the faculty members and the existing resources to run calculations larger than could be run if the same grant money were used to simply put a new system on campus. This can result in getting access to a very large amount of computing resources, as a few hundred thousand dollars buys a large amount of computing power at today's prices.

Commercial Queue: Available to industry customers. These customers are paying by the dedicated hour for access to the computing resources. For a quote on purchasing processing time, contact the HPC staff at **hpc@asc.edu** (mailto:hpc@asc.edu)

Sysadmin Queue: Used by the staff at the Alabama Supercomputer Center. It is used for testing new queue settings, reproducing problems users are having, testing hardware, and other administrative functions.

NOTE: If you type in a time limit for the job, it must be at least 1 hour.

Monitoring Jobs

Although the queue system will allocate resources when readily available, you can monitor the state of any job you have submitted.

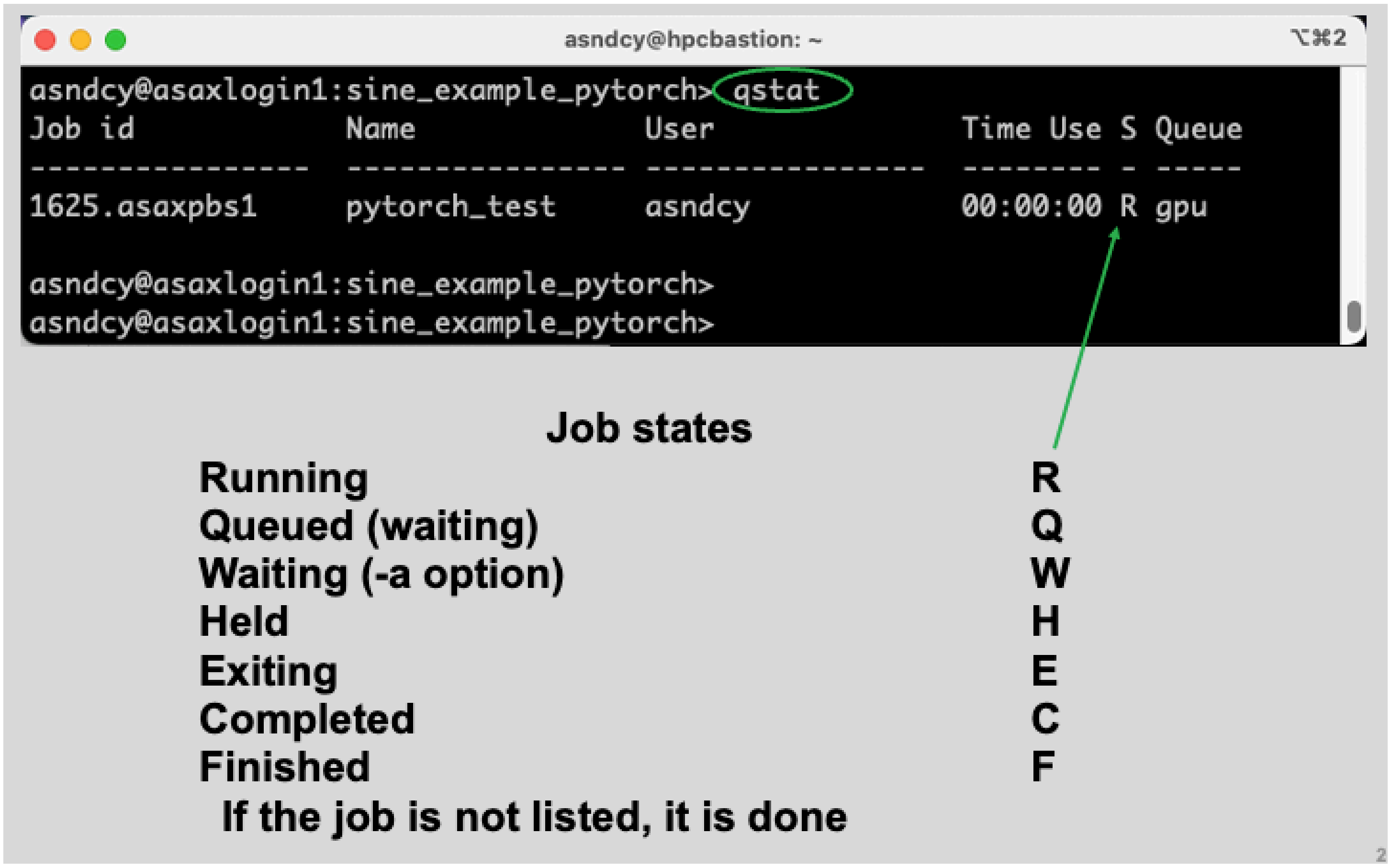
The "qstat" command shows what jobs are running and pending in the queue system. The qstat command supports a wide range of display options. There are convenience scripts for some of these that are named "qstat2", "qstat3", etc.

```
Job id           Name             User             Time Use S Queue
---------------- ---------------- ---------------- -------- - -----
4633.asaxpbs1    Npp3h2o9215ac1b* ualfrd001               0 Q large
17451.asaxpbs1   hopper           aubdbg001               0 Q gpu
17547.asaxpbs1   runmodelshSCRIP* aubsbw001               0 Q bigmem
17904.asaxpbs1   eso3bgjfG16      aubmxb002        562:14:* R large
17977.asaxpbs1   10               usaabp           164:27:* R medium
18062.asaxpbs1   submitprocjobss* aubkzc           00:00:05 R small
```

In the example above, the asterisk (*) indicates the information is wider than the column.

The first two columns of this output shows a list of jobs. The job number is in the left hand column. The job number can be used to get more information about the job, kill the job, or request help from the ASC staff.

This output also show a job status indicated by R or Q. If the status is R the job is running. If the status is Q the job is waiting to run, or queued. Other qstat options can be seen with the command "man qstat". To use these options, it may be necessary to turn off the default behavior with the command “unalias qstat”.



```
asndcy@asaxlogin1:sine_example_pytorch> qstat
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
1625.asaxpbs1     pytorch_test     asndcy            00:00:00 R gpu

asndcy@asaxlogin1:sine_example_pytorch>
asndcy@asaxlogin1:sine_example_pytorch>
```

Job states

Running	**R**
Queued (waiting)	**Q**
Waiting (-a option)	**W**
Held	**H**
Exiting	**E**
Completed	**C**
Finished	**F**

If the job is not listed, it is done

Tip: The staff at the Alabama Supercomputer Center have written a number of scripts to show what your jobs are doing in more convenient formats. Jobs may be pending for a number of reasons. The job could be requesting more memory, CPUs, or software licenses than are presently available. It is sometimes the case that the requested resources aren’t within the capabilities of the cluster. These scripts are named qstat2, qstat3, etc.

Once a queued job completes, an additional file will be created in the directory with the job inputs. This is referred to as an error log file. The file name consists of the job name from the queue and the queue job number. This file contains information about how the job was submitted to the queue, stdout output from the job, and stderr output from the job. If a job fails to start or fails to run to completion, this file is one of the primary places to find out what is wrong. If you contact the ASC staff for help, they will want to see this file. Simply giving the technical staff the directory you are in and the job number is sufficient. The technical staff members can go into user directories, but do so only when asked for help.

The command “**jobinfo -j JOBNUMBER**” gives information about the job. If this command is called before the job is complete, it gives information about how the job was submitted to the queue. Calling the jobinfo command after the job is complete shows information about the jobs resource utilization, like this;

```
asnrye@asaxlogin1:tests> jobinfo -j 1616
####################################################################
# Your username for this job is:                 asnrye
# Your group for this job is:                    null
# Your job ID is:                                1616.asaxpbs1
# Your job name is:                              ExampleJob
# Your job queue is:                             small
# Your job ran on nodes:                         asax001/0*2
# Your number of processors used:                2
# Your job work directory was:                   /home/asnrye
# Your job was submitted at:                     Fri Oct 6 16:18:33 2023
# Your job started at:                           Fri Oct 6 16:18:33 2023
# Your job ended at:                             Fri Oct 6 16:21:15 2023
# Your job elapsed time is:                      00:02:42
# Your job dedicated time is:                    00:05:24
# Your job cpu time is:                          00:00:00
# Your job was last modified:                    Fri Oct 6 16:21:17 2023
# Your requested wall time is:                   01:00:00
# Your job cpu parallel efficiency is:           0.00%
# Your requested memory is:                      2gb
# Your max memory used:                          2800kb
# Your job memory efficiency is:                 0.14%
# Your job state is:                             F
# Your job exit code is:                         0
# Your requested resource was:                   1:ncpus=2:cpuchip=Milan
# Your job commercial value is:                  $ 0.0072
####################################################################
```

This information is useful for determining why the job died, and what resources it needs. For example, if the job requested 2gb of memory and it used 2234mb (>2gb) of memory, the queue system may have killed the job due to exceeding its memory allocation. The results from a job that ran correctly can be used to determine how much memory should be requested next time.

Time is displayed in the format **DAYS-HH:MM:SS**. A memory format like **500Mc** means 500 megabytes per core, so the user in this example requested 2 cores and 1 GB of memory.

If you are having issues with a job hanging in the queue for whatever reason, a good command to try is **reason_pending**. This script gives useful information for debugging stuck jobs.

Tip: Jobs that request more CPUs and memory can take longer to get into a run state. Thus it is to the users advantage to request a reasonable amount of memory, usually about 20% more than the job is expected to need. Choosing the optimal number of CPUs is discussed in the parallel processing section of this manual.

Deleting Queued Jobs

It is sometimes necessary to delete jobs from the queue. This can be because the job was submitted with the wrong inputs, isn't running correctly, or is stuck in a pending state due to invalid queue settings. Running or pending jobs can be deleted with the command

```
qdel <job_number>
qdel -Wforce <job_number>
```

Those can be jobs are in a run state (job state R), but there aren't actually processes running on the compute nodes. These are called zombie jobs. It can be a queue system bug. It can be a software bug where the MPI master process exited but one of the slave processes is still running. It can come from other types of network latency issues. If you encounter a job like this

Step 1) try to kill them with

```
qdel JOB_NUMBER
```

Step 2) try to kill them with

```
qdel -Wforce JOB_NUMBER
```

Step 3) email **hpc@asc.edu** (mailto:hpc@asc.edu) a message with the job number and saying you can't delete the job.

Running Existing Applications Software

Applications software installed by the ASC staff may have both an associated queue script and documentation on this website. In the example of the Gaussian software, a calculation using the input file water.com (which would have to be in the current directory) can be run with the following command

```
rungaussian water.com
```

The rungaussian script will display a list of queues available to you, ask you to type in the name of a queue, number of processors, time limit, memory, and job name.

All of the queue scripts on the system use this same set of prompts for the queue, memory, etc. If it is not possible for a given program to run in parallel, the script will not ask for a number of CPUs. With many programs, it is necessary to construct the inputs appropriately for parallel execution, as well as requesting multiple CPUs from the queue script. It is possible to respond to all of the interactive prompts by pressing "Return" to take the defaults. Once the queue has been entered, the default prompts will reflect reasonable values for that queue.

Tip: If you use the same queue settings every time, you can set preferences so it knows what you want and doesn't give the prompt. This is done by editing the .asc_queue file in your home directory. The comments in that file make it self-explanatory.

Running User Written Software

When the user has written their own software, there will not be a queue script already available tailored to that software. There is however, a queue script named "run_script" which is configured to run any script that does not require arguments. run_script forces the job to use one or more processors, all on a single node. There is a "run_script_mpi" command for software capable of using processors on different nodes. Consider the example of a user that has compiled a program on ASAX. For example, the program may be named foo and require an argument with the name of an input file such as bar.inp In this example, the program can't be submitted directly to run_script because it requires an argument. However, the user can create a script, say named myscript, that contains the following text.

```
#!/bin/bash
# script to run the foo program on asax
source /apps/profiles/modules_asax.sh.dyn
module load gcc/11.3.0
./foo bar.inp
```

The user can put this text in the myscript file using a text editor such as nano. The characters "./" in front of the program name indicate that the script expects to find the foo program in the same directory as the bar.inp and myscript files. The source and module lines load tell your program how to find the libraries that came with version 15.0.0 of the Intel Compilers. If you had to load modules to compile the software, you often must load the same modules to run that software. See the chapter on Using Modules for more details.

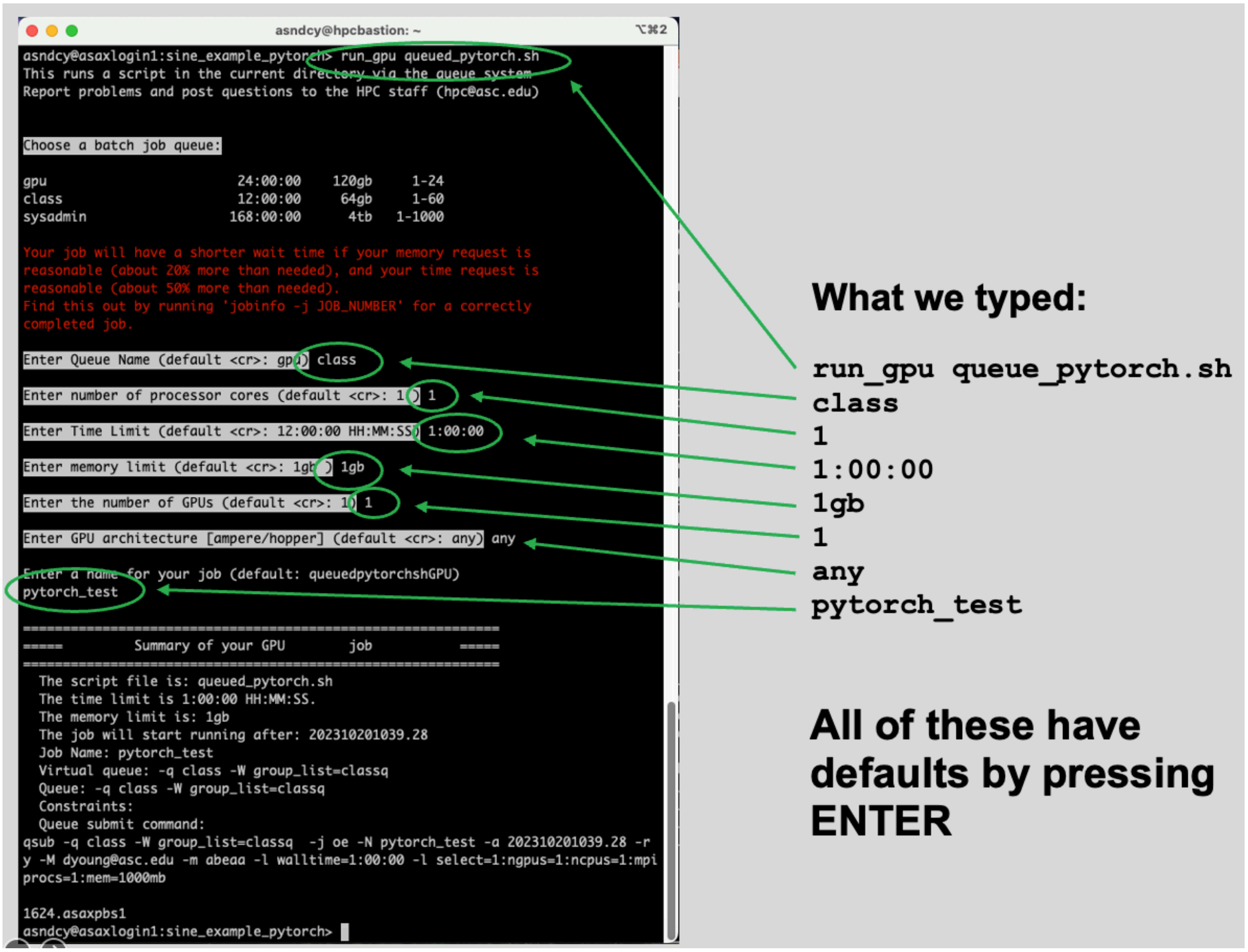
The myscript file must be made executable with a command like this

```
chmod +x myscript
```

This calculation can now be submitted to the queue system with the following command;

```
    run_script myscript
```

The run_script program will give the same prompts as the other queue scripts with one additional prompt. The final prompt asks which cluster it should be allowed to run on (typing Return to take the default is fine).



```
asndcy@asaxlogin1:sine_example_pytorch> run_gpu queued_pytorch.sh
This runs a script in the current directory via the queue system
Report problems and post questions to the HPC staff (hpc@asc.edu)


Choose a batch job queue:

gpu                        24:00:00    120gb     1-24
class                      12:00:00     64gb     1-60
sysadmin                  168:00:00      4tb   1-1000

Your job will have a shorter wait time if your memory request is
reasonable (about 20% more than needed), and your time request is
reasonable (about 50% more than needed).
Find this out by running 'jobinfo -j JOB_NUMBER' for a correctly
completed job.

Enter Queue Name (default <cr>: gpu) class

Enter number of processor cores (default <cr>: 1 ) 1

Enter Time Limit (default <cr>: 12:00:00 HH:MM:SS) 1:00:00

Enter memory limit (default <cr>: 1gb ) 1gb

Enter the number of GPUs (default <cr>: 1) 1

Enter GPU architecture [ampere/hopper] (default <cr>: any) any

Enter a name for your job (default: queuedpytorchshGPU)
pytorch_test

============================================================
=====           Summary of your GPU        job         =====
============================================================
  The script file is: queued_pytorch.sh
  The time limit is 1:00:00 HH:MM:SS.
  The memory limit is: 1gb
  The job will start running after: 202310201039.28
  Job Name: pytorch_test
  Virtual queue: -q class -W group_list=classq
  Queue: -q class -W group_list=classq
  Constraints:
  Queue submit command:
qsub -q class -W group_list=classq  -j oe -N pytorch_test -a 202310201039.28 -r
y -M dyoung@asc.edu -m abeaa -l walltime=1:00:00 -l select=1:ngpus=1:ncpus=1:mpi
procs=1:mem=1000mb

1624.asaxpbs1
asndcy@asaxlogin1:sine_example_pytorch>
```

Now consider what could be done if the program foo has been compiled twice to make both an AVX executable and a AVX-512 executable. In this case, the script must be able to identify whether it is running on a node that supports AVX-512 instructions and use the correct version of foo. A script like this is shown below.

The following script must determine whether it is on an AVX-512 node or an older compute node.

```
     #!/bin/bash
     #  script to run foo on either an AVX-512 or older node
```

```
source /apps/profiles/modules_asax.sh.dyn
avx512_check=$( cat /proc/cpuinfo | grep avx512 | wc -l )

# see if I'm on an AVX-512 node
if [ "$avx512_check" -gt "0" ]
then

  module load intel/18.0.3.222

  foo_path=/home/asndcy/foo/bin_avx512
else

  module load pgi/18.5

  foo_path=/home/asndcy/foo/bin_other
fi

# see if foo_path has been set
if [ $foo_path ]
then

  # run the program

  ${foo_path}/foo bar.inp
else

  echo "ERROR: unrecognized node type"
fi
```

For more information on writing scripts like this, see the section of this website on Shell Scripts on the Linux page.

This new script can be made executable with the command "**chmod +x SCRIPT_NAME**" then submitted to the queue with run_script. This time, when run_script asks which cluster to use, the user can press “Return” to signify that it can run on either cluster.

Another function typically found in scripts submitted to the queue is copying the work over to be done on the /scratch drive. The user home directories are visible to all compute nodes, but these directories are on a lower performance file system. If the calculation does a large amount of accessing data on the disk drive, it will run faster on the high performance /scratch file system. Here is an example of a script that utilizes /scratch

```
#!/bin/bash
#
#  Sample script for submitting jobs to the queue system
#  This script shows how to run the calculation
#  in the /scratch directory
#
#  Replace the USER name in this script "asndcy"
#  with your own user name.
#  Replace the program name "mandy_gcc" with your own program name.
#  Replace the input file name "test1.in" with your own input name.
#
#  This script must be made executable like this
#    chmod +x my_script
#
#  Submit this script to the queue with a command like this
#    run_script my_script

#  put in my user name
USER=asndcy

# create a directory on /scratch
mkdir /scratch/$USER

#  copy the program and input files to scratch
cp mandy_gcc /scratch/$USER
cp test1.in /scratch/$USER

# run the program
cd /scratch/$USER
./mandy_gcc test1

# copy the results back to my home directory
cp test1.bmp /home/$USER
```

The application specific queue scripts and the run_script & run_script_mpi commands are specific to the Alabama Supercomputer Center. Everything that you can do with the queue system is accessible through these interfaces. However, some users may be more familiar using “qsub” as is done at many other computing centers. The use of run_script and the application queue scripts is recommended. The use of qsub directly is strongly discouraged, and not supported by the HPC technical staff. The run scripts wrap qsub, and thus allow improvements and adaptations to changing needs on a regular basis. Thus users who want to use qsub directly will find that they have to do so without documentation or technical support, and that they will have to change the options they give to qsub as frequently as every six months.

User written MPI software should be launched with the **mpiexec** command when run through the queues. Short tests (no more than 2 processors and 10 minutes) can be run on the login nodes with the **mpirun** command. See the MPI documentation on this website for additional information.

There are a number of variations on the run_script command. Here is a summary.

Script name	what it does
run_script	run a job using 1 or more processor cores on a single node, and ask where to run it
run_script_mpi	run a job using 1 or more processor cores across multiple nodes, and ask where to run it
run_script_milan	run a job using 1 or more processor cores on a single node with an AMD Milan processor
run_script_intel	run a job using 1 or more processor cores on a single node with an Intel processor
run_script_omp	equivalent to run_script
run_script_8p	run a multi-node job that uses 8 processor on each node
run_script_mpi_416	Runs a 4 node job with 16 cores on each node.
run_script_scratch	create a temporary directory in the /scratch file system, copy everything from the current directory to /scratch, do the calculation on /scratch, copy everything from the temporary directory back to the current directory, delete the temporary directory. The current directory must be a subdirectory inside of your home directory. Your script should specify relative paths only. All processors will be on the same node.
run_script_local	create a temporary directory in the /scratch-local file system, copy everything from the current directory to /scratch-local, do the calculation on /scratch-local, copy everything from the temporary directory back to the current directory, delete the temporary directory. The current directory must be a subdirectory inside of your home directory. Your script should

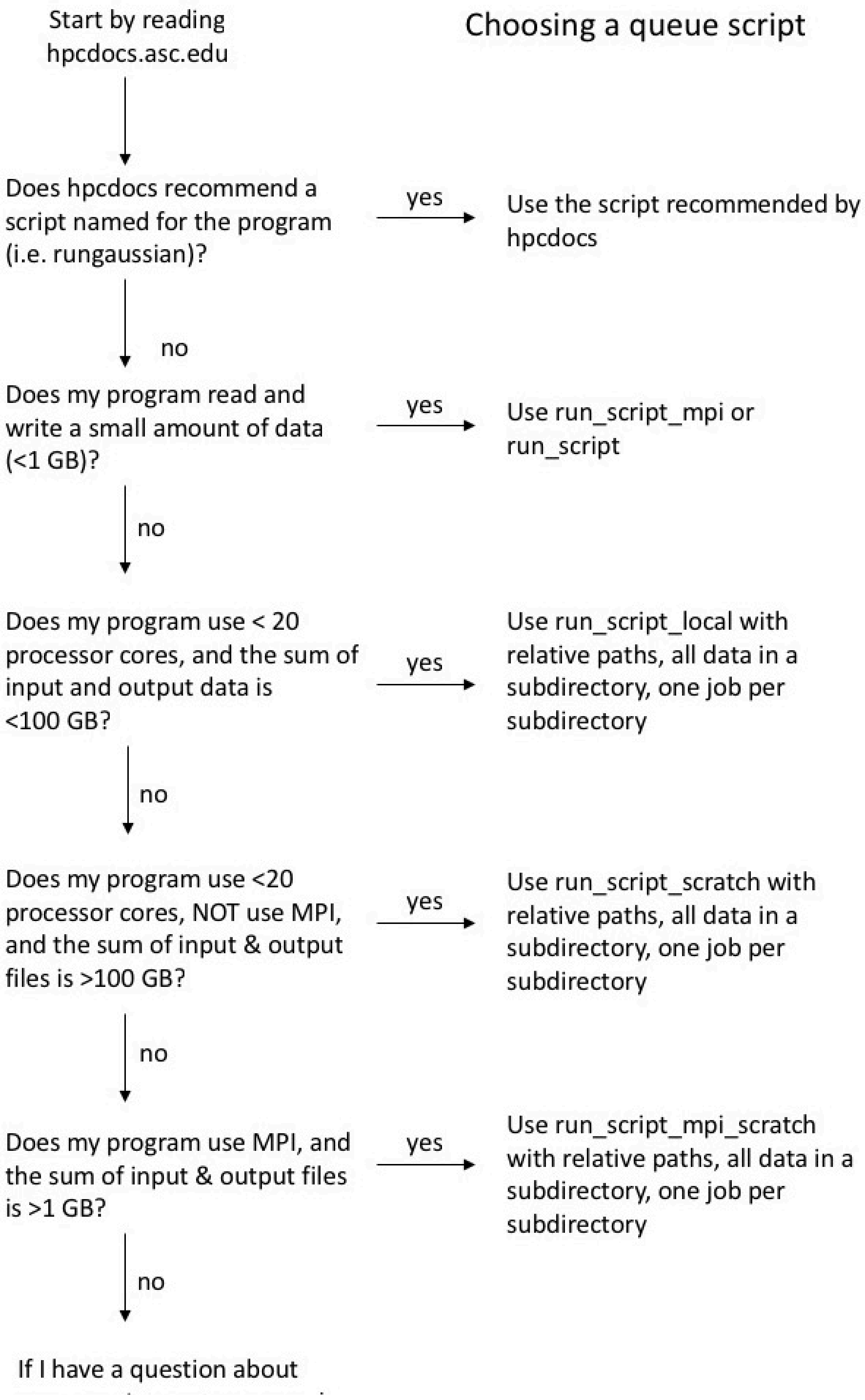
	specify relative paths only. All processors will be on the same node.
run_script_mpi_scratch	create a temporary directory in the /scratch file system, copy everything from the current directory to /scratch, do the calculation on /scratch, copy everything from the temporary directory back to the current directory, delete the temporary directory. The current directory must be a subdirectory inside of your home directory. Your script should specify relative paths only. Processors may be on different nodes of the same architecture.

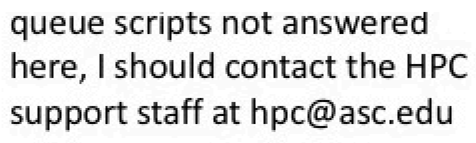
Choosing The Best Queue Script

There are three styles of queue scripts in use on the ASAX.

1. Some of the scripts, such as rungaussian, are specific to one software package. These scripts stage large files in the location that gives the fastest execution of the program, often in the /scratch or /scratch-local directories.
2. Some of the scripts are made to run a user written script through the queue system. Many of these such as run_script and run_script_mpi run the calculation from the directory where you submitted it. This lets the user control whether the calculation runs in their home directory, or if they choose to stage files on the /scratch file system and run the calculation there. Prior to 2019, the home and scratch file systems had similar performance so scratch was only needed for staging very large files. System upgrades in 2019 have given scratch much higher performance than home. Thus it may be to the users advantage to run calculations that could be done in their home directory on scratch to get a shorter execution time.
3. The last category of scripts are ones that are designed to run from a subdirectory of your home directory, then automatically copy files from there to scratch, do the calculation in scratch, then copy all of the files (both input and output) back. This can be both convenient and give shorter execution times, but it has implications as discussed below. Scripts with this behavior include run_script_scratch, run_script_local and run_script_mpi_scratch

Here is a decision tree to aid in choosing.


Choosing a queue script
Start by reading hpcdocs.asc.edu
Does hpcdocs recommend a script named for the program (i.e. rungaussian)?
yes
Use the script recommended by hpcdocs
no
Does my program read and write a small amount of data (<1 GB)?
yes
Use run_script_mpi or run_script
no
Does my program use < 20 processor cores, and the sum of input and output data is <100 GB?
yes
Use run_script_local with relative paths, all data in a subdirectory, one job per subdirectory
no
Does my program use <20 processor cores, NOT use MPI, and the sum of input & output files is >100 GB?
yes
Use run_script_scratch with relative paths, all data in a subdirectory, one job per subdirectory
no
Does my program use MPI, and the sum of input & output files is >1 GB?
yes
Use run_script_mpi_scratch with relative paths, all data in a subdirectory, one job per subdirectory
no
If I have a question about


queue scripts not answered
here, I should contact the HPC
support staff at hpc@asc.edu

Here are more things to consider when selecting a queue script.

If there is a script provided specifically for a piece of software, it is best to use that unless you have a solid reason why it won't work for you needs.

Scripts that have "mpi" in the name allow the software to run across multiple nodes on the network. This only works if the software is parallelized with the MPI library and launched with an mpi launcher such as mpiexec. If the name of the script, does not have "mpi" in it the queue system will give it all processors on the same node.

Software doesn't use multiple processors unless someone programmed it to do so. If the software doesn't explicitly give a way do this (by launching an MPI job, setting OMP_NUM_THREADS, or a -threads flag), assume that the software can only use one processor and only request one from the queue script. In this case, our convention is to recommend a queue script without "mpi" in the name.

Consider the scripts that copy the directory contents to scratch, do the calculation, then copy everything back. If the input data is really big, that can be a bunch of extra data moving that takes time. If you run two such jobs from the same directory, job 1 might be copying the input data back while job 2 is trying to copy the input data to scratch, and thus they could collide killing both jobs. A safe format is to run job 1, wait until it is complete, then submit job 2 from the same directory. Or create two duplicate subdirectories with the same input data, provided that the data isn't so large that duplicates start eating significantly into your home directory disk quota.

run_script_scratch, run_script_local, and run_script_mpi_scratch aren't smart enough to copy only the output files back to your home directory. They copy everything back, even though that means copying input data back onto itself. They do this to ensure nothing is lost. However, you can make your own script that makes a directory in /scratch, copies input data to that directory, runs the job there, then copies only the output files back, then deletes the directory on /scratch. This smarter script can be submitted to the queue with run_script or run_script_mpi.

The scratch file system is higher performance than the home file system. This makes a big difference if the software makes tens of thousands of little files (i.e. Trinity). It also makes a big difference if the software makes one big file then does many random reads on it (i.e. the Gaussian integral files). The difference between the two file systems is not as big if there is one large input file which is read once, linearly from beginning to end (some bioinformatics programs work this way).

New students often point out that they only know the name of the software their adviser suggested using, and not how it accesses files. This lower level understanding of what the software is doing inside is part of the education that needs to be obtained by someone getting a graduate degree in any field that utilizes large computer systems. Other key topics for a researcher to understand include time complexity to predict how long your calculations will take to run (typically taught in the data structures and algorithms course in the computer science curriculum, but described on this documentation website as well) and numerical analysis from the mathematics curriculum which gives you an understanding of how accurate the results may be.

Queue System Fairness

Job queue systems are highly configurable. Even another facility using PBS may have it configured to operate much differently from the PBS installation at the Alabama Supercomputer Center. An organization that does weather prediction may have the queue system configured with reservations to ensure processors are available to run todays weather predictions at the same time every day. Many organizations have a system in place to charge users for the cost of processing time. Some organizations allocate computer resources based on the relative priority of various projects. The administration of the HPC systems at the Alabama Supercomputer Center is based on the following management principles;

- Select hardware to minimize unplanned outages, meet users' needs, and get a good amount of processing power for the cost.
- Provide some resources not available on most campuses (i.e. very large memory systems)
- Provide access to cutting edge technologies, as budget allows.
- Put significant effort into good documentation, in order to operate with a minimal support staff.
- Proactively innovate for ease of use.
- Proactively correct problems.
- ASA does not render judgment on the merit of client work.
- No in-house research is being done at the Alabama Supercomputer Center.
- Make the systems useful to as many clients (and disciplines) as practical.
- Assign resources based on policies that are applied equally to all users in the same category (research, classroom instruction, ASA's collaborators, or commercial).
- Encode policies in queue software, as much as possible. We do not manage based on "gentleman's agreements" or daily/weekly monitoring of users' behavior.
- Have a quick turnaround time on user help requests, with a necessarily longer turnaround time on software install requests or purchase requests.
- Find a balance of three, sometimes conflicting, goals;

1. Maximize allocation of resources (processors in a run state).
2. Attempt to give users submitting large job loads an approximately equal number of processor cores as other big users.

3. Set priorities to give shorter average queue wait times for users submitting a light job load than for users submitting a heavy job load.

In order to implement these ideals, a quantified fairness metric is used. Each month, the following ratio is computed for all users. This gives a range of values, typically arranged as a Gaussian distribution with some users over- or under-advantaged and many clustered around the mean. Queue system scheduling configuration settings are then adjusted to minimize the standard deviation of this distribution.

$$\frac{\text{(monthly dedicated hours for person X)}}{\text{(average wait time in seconds for person X)}} = \text{user fairness for X}$$

We will note that there are some implications of this fairness metric. All users are considered equal. Thus job scheduling does not take into account the research group, project, or academic department. Jobs requesting many processors should and do wait in a pending state longer than single processor jobs, as the multi-processor jobs will get more processing time once they get into a run state. Some jobs may wait a shorter or longer than average amount of time, as fairness is defined on average but not on a per-job basis.

Tip: The users who are most often under advantaged in fairness are users who are requesting far more memory or processing time than their job needs.

The queue system is configured to view the hardware as a large pool of available resources. As such, there are not X many processors assigned to the small queue, and Y many assigned to the large queue. However, class queue jobs can preempt regular queue jobs to ensure that homework assignments do not wait behind week-long research calculations. Preemption chooses the smallest, most recently started, job then puts it back into a pending state to run later. Likewise, non-GPU jobs can use all of the cores on the GPU nodes if no GPU jobs are pending, but a non-GPU job can be preempted once a GPU is needed. Overall, preemption is a big win because for every job that gets preempted another hundred jobs ran hours sooner than they would have if the resources were reserved for class and GPU jobs.

Frequently Asked Questions about PBS

Q: Why was my job canceled by preemption?

A: When a class homework assignment is submitted to the class queue, those jobs get into a run state immediately so that homework assignments don't wait behind a big research calculation. If necessary, the homework jobs will preempt the smallest research calculation that has most recently entered a run state. The research calculation is put back into a pending state to run later. This also happens when there is a hardware error on the compute node where your job is running.

Prior to implementing preemption, 100-200 processor cores were reserved for homework jobs. Those processors sat idle 95% of the time waiting for a homework assignment to be submitted. Preemption allows the resources to be utilized more fully. Statistically, for every job that gets preempted and run a few hours later another fifty jobs got into a run state hours sooner.

Q: Can PBS use qsub commands or #PBS directives that work on a different computer system?

A: Probably not. Queue systems are highly configurable. Another facility that uses even the same version of PBS may have chosen configuration options that require a completely different set of command line flags

It is strongly recommended that you use the "run" scripts such as "runamber" provided on the Alabama Supercomputer Center systems to run submit your work to the queues.

Q: WHERE CAN I FIND MORE DOCUMENTATION?

A: Some of the commands have man pages. For example, you can get more information about the qstat command by typing "man qstat".

PBS documentation is available at:
https://help.altair.com/2022.1.0/PBS%20Professional/PBSUserGuide2022.1.pdf (https://help.altair.com/2022.1.0/PBS%20Professional/PBSUserGuide2022.1.pdf)

Q: WHY IS MY JOB QUEUED BUT NOT RUNNING?

A: There is a "reason_pending" command to find out why a job isn't running.

```
reason_pending my_job_number
```

or

```
checkjob_asn my_job_number
```

Where the job number is the number seen at the left side of the qstat display.

Q: How many jobs can I submit at once?

A: There is a limit to how many jobs the queue system can simultaneously handle this is based on the memory and capabilities of the queue system server. That limit for the entire system (over 1000 users) is currently 10,000 jobs. At any one time, there is little reason to submit more than 1,000 jobs since only a tenth of those will be able to get into a run state. Limits per user and per queue may get adjusted on short notice if your usage impacts other users.

Q: What queues are available?

A: Type "qlimits" to see a list of queues along with the amount of memory, number of processors, and time limits for those queues.

INCORPORATE THE FOLLOWING WORDING. The class queue has a much shorter time limit than the research queues because homework assignments are supposed to be small examples that illustrate the principles involved in research, without the scale of research projects.

Q: Does the queue limit the number of jobs that can be in a run state?

A: The special queue allows 1 job at a time per user in a run state. The benchmark queue allows 1 job at a time per user in a run state. The class queue allows a maximum of 5 jobs at a time in a run state per user. The number of GPU jobs in a run state is usually limited by the availability of GPU chips, but is not limited by the queue system. The standard queues like the large queue do not have a limit on the number of jobs, but are limited to a maximum of 200 cores per user in a run state.

Q: What is the benchmark queue, and why do jobs there wait so long to get in a run state?

A: See the **benchmark queue documentation** (https://hpcdocs.asc.edu/content/benchmark-queue)

Q: How do I run jobs larger than the large queue limits?

A: See the **special queue documentation** (https://hpcdocs.asc.edu/content/special-queue)

Q: How do I change the default settings in the queue submit command.

A: Edit the .asc_queue file in your home directory.

Q: My software ran on the login node, but not through the queue. What is wrong?

A: The PBS queue system is more sensitive to your login environment than other queue systems. Some problems can be fixed by;

1. Run "module purge" prior to submitting your job.
2. Remove module load commands from your .bashrc.local and .bashrc.local.asax files.

Q: My job was running, but now it is pending again. What happened?

A: Jobs can be requeued if the job was preempted by a class or gpu job.

Q: How long should I expect a job to wait in the queue system.

A: The average wait time is currently about 30 hours (not bad compared to 3-4 days at NCSA or 1-2 weeks for academic jobs at Oak Ridge). Single processor jobs get into a run state a bit quicker. Jobs requiring many processor cores wait longer.

Q: What can I do to get it into a run state more quickly?

A: Here are strategies for getting your job into a run state more quickly.

- If you are choosing between 4, 8, or 16 CPUs, the 4 CPU job will almost always get in a run state first. One viable strategy is always using the fewest number of cores that will let the job complete within the 15 day wall clock time limit.
- Another valid work flow is to submit a 16 core job, if it doesn't get in a run state in ten minutes cancel and resubmit as 8 cores, wait 10 mintues, etc.
- In general the lightest load on the system is on Monday morning, and the heaviest load Friday evening. Thus jobs will get in a run state more quickly if the ones requesting the most cores are submitted early Monday morning and smaller jobs are submitted later in the week.
- There are almost never 64 cores available, so you should expect jobs of that size to wait fairly long regardless of when you submit them.
- Request a reasonable amount of memory and reasonable run time for you job.
- If running MPI jobs on ASAX, use run_script_mpi not run_script.
- There is also the option of purchasing exclusive use of some compute nodes for your research group if you are looking to spend a few hundred thousand dollars.

Q: How do I maximize productivity (the total amount of work completed in a year).

A: Keep enough jobs in the queue to fill about 250 processor cores. This way you always have a pending job ready to use processor cores when they become available.

Q: Why does the error log file from my queued job have messages about singularity.

A: Some software is accessed via containers such as Singularity or Docker. Containers are similar to virtual machines. If the software can't run on this computer which uses Rocky Linux, it might run in a container which runs the software in an Ubuntu Linux environment.

Q: What happens if I request 2 processor cores from run_script, but wrote my script to use 4 processor cores?

A: Several things might happen.

- Your software may not run at all.
- If your software does run, it will probably run slower.
- If your software really tries to heavily use more processor cores than you requested, you and your research adviser will get a sternly worded email from the HPC technical staff threatening to permanently ban you from using the supercomputers.

Related Weblink:

More information can be found at https://help.altair.com/2022.1.0/PBS%20Professional/PBSUserGuide2022.1.pdf (https://altair.com/pbs-professional)