Working with the Queue System at the Alabama Supercomputer Center

Nearly all supercomputing facilities like the Alabama Supercomputer Center (ASC) use a job queue system. A job queue system is similar to a printer queue; a pile of work can be submitted and the queue system software will schedule and start each job when the necessary resources become available.

In the case of a job queue system, the resources being managed are computer processors, memory, and sometimes software licenses.

The queue system is the researcher's friend. If you want to get a large amount of work done, the best thing you can do is learn how to utilize a queue system effectively.

The queue system also guarantees that the user's job will receive the number of CPUs and amount of memory that they requested when the job was submitted.

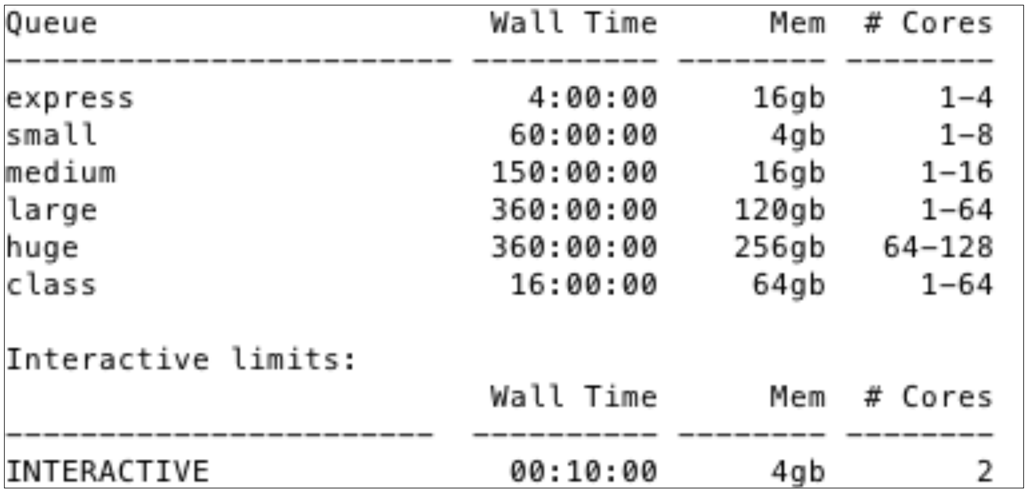
Before being able to access the queue system, one first needs to **connect to the ASC**. This is done from your local machine using a secure shell (SSH) in a Terminal emulator (such as Terminal on Apple OS X or SecureCRT on Windows):

```
ssh username@uv.asc.edu
```

Once logged into the ASC, there are a number of queues that are available depending on the type of job you plan to run. A list of the available queues can be displayed with the

```
qlimits
```

command. The `qlimits` command is called without any arguments. 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     1-64
huge                       360:00:00     256gb   64-128
class                       16:00:00      64gb     1-64

Interactive limits:
                           Wall Time       Mem   # Cores
------------------------  ----------- -------- --------
INTERACTIVE                 00:10:00       4gb        2
```

The "Wall Time" column in this output gives the amount of time, per CPU, 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 CPUs can be requested by a job in that queue

Submitting jobs to the Alabama Supercomputer Center queues

When the user has decided on the analyses to be performed, those specific commands and their parameters are placed into a text file generically called a "script". Such scripts are submitted to the queue using the "`run_script`" command, which is configured to run a script that does not require arguments. The `run_script` command forces the job to use one or more processors, all on a single node. There is also a "`run_script_mpi`" command for software capable of using processors on different nodes.

For example, a program that we want to execute might be named "`Trinity.pl`" and requires arguments with the name of input files such as `test_left.fq` and `test_right.fq.` In this example, the program can't be submitted directly to `run_script` because it has required arguments. However, the user can create a script, named `Trinity_job.sh`, that contains the following information:

```
#!/bin/sh
# script to run Trinity.pl (THIS IS A COMMENT ON THE SCRIPT)
source /opt/asn/etc/asn-bash-profiles-special/modules.sh
module load intel/13.0
./Trinity.pl --left test_left.fq --right test_right.fq
```

The user can put this text in the `Trinity_job.sh` script using a text editor such as `nano`. The characters "./" in front of the program name indicate that the script expects to find the `Trinity.pl` program in the same directory as the input files and script itself. The `source` and `module load` sections tell your program how to find the libraries that came with version 13.0 of the Intel Compiler. Notably, if you had to `load modules` to compile the software into a binary in the first place, you must load the same modules to run that software. We will discuss what your source and module lines should look like for the different programs you may want to use.

The script file must be made executable (for the system to be able to run it) with the command:

```
chmod +x myscript (WHERE "MYSCRIPT" IS SPECIFIC TO THE SCRIPT YOU WROTE)
```

The job/analysis can now be submitted to the queue system with the following command:

```
run_script myscript
```

After executing `run_script myscript` you will see several prompts about the job that you will need to answer:

```
Choose a batch job queue:
Queue              CPU          Mem         # CPUs
--------------     ----------   ------      ------
```

```
small-serial       40:00:00     4gb          1
medium-serial      90:00:00     16gb         1
large-serial       240:00:00    120gb        1
small-parallel     48:00:00     8gb          2-8
medium-parallel    100:00:00    32gb         2-16
med-parallel       100:00:00    32gb         2-16
large-parallel     240:00:00    120gb        2-64
daytime            4:00:00      16gb         1-4
express            01:00:00     500mb        1

Enter Queue Name (default <cr>: small-serial) small-parallel

Enter number of processor cores (default <cr>: 2 ) 2

Enter Time Limit (default <cr>: 96:00:00 HH:MM:SS) 24:00:00

Enter memory limit (default <cr>: 1gb ) 2gb

Should this job run on uv, or dmc (default: any) dmc

Choose your job starting date and time (<cr> for now):

If not running right now, enter time and date as
[[CC]YY]MMDDhhmm[.ss]

Enter a name for your job (default: watercomG09)
Trinity_test

============================================================
=====              Summary of your script job          =====
============================================================
The script file is: Trinity
The time limit is 24:00:00 HH:MM:SS.
The memory limit is: 2gb
The number of CPUs is: 2
The job will start running after: 200810131149.36
Look for: Trinity_test in queue: small-parallel

Job number 82641.mds1.asc.edu
```

Example responses are in ***italics/bold***

Pressing enter will automatically choose the default values

These options may not be available depending on the queue

NOTE: In the Bioinformatics Bootcamp you will be using the ***class*** queue. This has limits of 64GB memory, up to 64 CPUs. The total amount of compute time per job will vary depending on the resources requested for it.

Checking the status of your job

The "`squeue`" and "`sjstat`" commands shows what jobs are currently running or pending in the queue system.

```
squeue
     This command gives basic output on queued jobs.
```

```
sjstat
     Also provides list of your currently queued jobs.
sjstat -v
     Verbose output of job listing.
sjstat -r
     Shows only currently running jobs, not failed or pending.
```

The first two of these commands show a list of jobs. `squeue` automatically displays only your jobs (e.g., you can't see all the jobs in the queue). The job number is in the left hand column. The numeric part of the job number can be used to get more information about the job, kill the job, or request help from the ASC staff. These commands 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.

One way to find out why a job isn't running is with the command:

```
jobinfo -j <job_number>
```

Once a job submitted to the queue completes, an additional file will be created in the directory were the job was started from. This is referred to as a 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 (i.e., what would have been otherwise printed to the screen) from the job, stderr output (i.e., any special errors that the program might have produced) from the job, and a listing of resource utilization. If a job fails to start or fails to run to successful completion, this file is one of the primary places to find out what is (or went) wrong. If you contact the ASC staff for help, they will want to see this file.
The resource utilization section of the log file looks like the following:

```
######################################################################
# Your job finished at : Mon Nov 8 13:10:39 CST 2010
# Your job requested :
cput=336:00:00,mem=2gb,neednodes=1:ppn=2,nodes=1:ppn=2,walltime=235:00:00
# Your job used :
cput=00:00:01,mem=4372kb,vmem=35580kb,walltime=00:00:32
# Your job's parallel cpu utilization : 1%
# Your job's memory utilization (mem) : 0.21%
# Alabama Supercomputer Center - PBS Epilogue
######################################################################
```

This entry is useful for a number of reasons, such as what level of computational resources were utilized or what might have lead to a job being "killed". 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.

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 deliberately terminated with the command:

```
scancel <job_number>
```

Use only the numeric part of the job number, not the .mds1 extension, when killing a job.