

Creating Job Submission Scripts with Grid Engine

December 2021

HPC using Grid Engine

Introduction

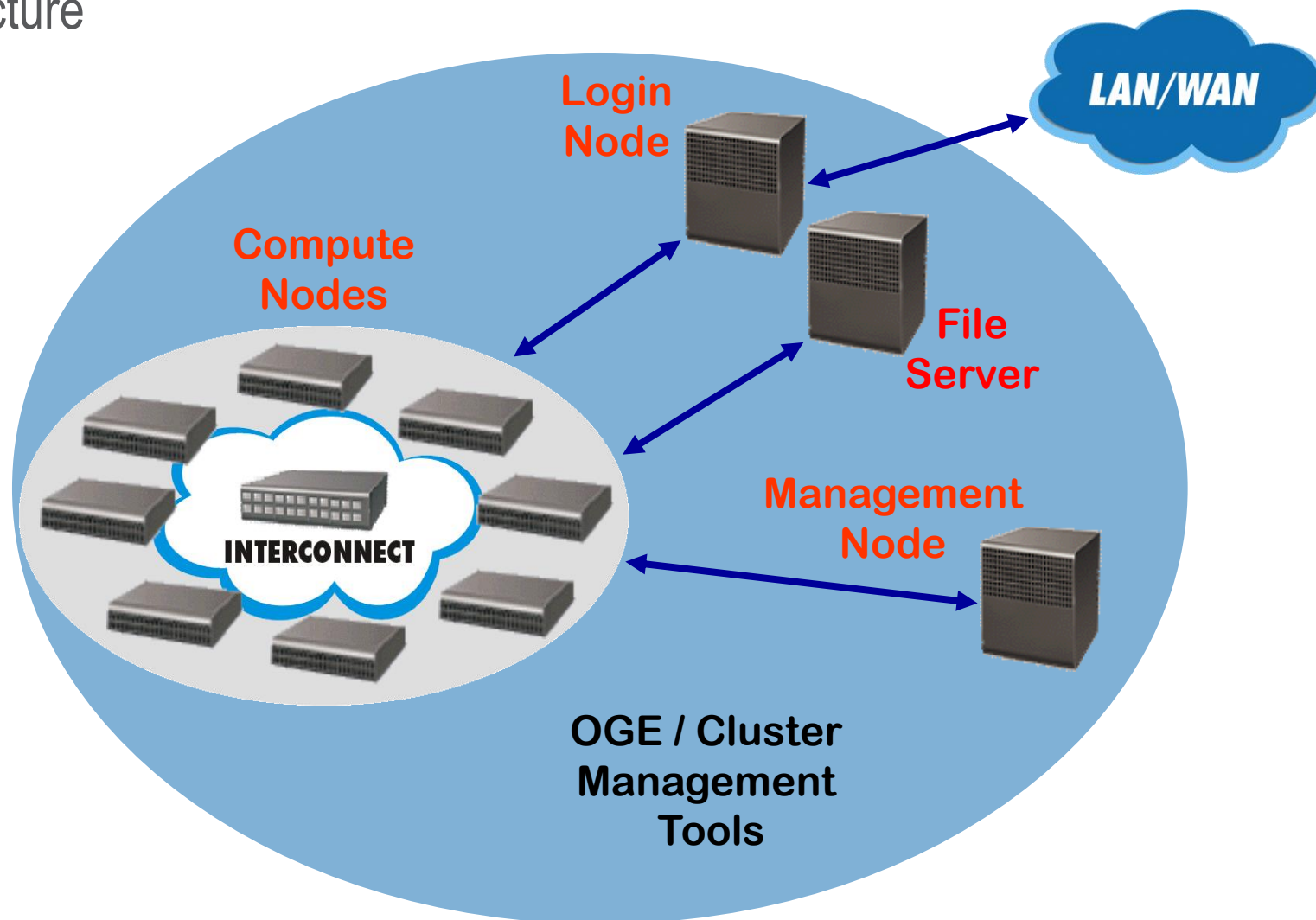
High-Performance Computing (HPC) resources are usually managed and accessed via a digital resource manager (DRM). One such DRM is Open Grid Engine (OGE, formerly Sun Grid Engine), which is employed on the Computer Science (CS) cluster. It is necessary to understand how to use OGE to make effective use of the cluster.

This course will cover the following topics:

1. HPC infrastructure.
2. Requesting resources – anatomy of a job script.
3. Managing jobs.
4. File management.
5. Common issues.

HPC using Grid Engine

HPC Infrastructure



HPC using Grid Engine

HPC Infrastructure

- Login nodes provide users access to the cluster, to manage files and data, and submit jobs. Users communicate with a login node from the command-line via an SSH session. A list of available login nodes is included with the welcome email sent when your cluster account was created. Please see hpc.cs.ucl.ac.uk/ssh-scp for more information on how to access a login node via SSH from outside the CS network.
- A basic understanding of Linux commands, including how to edit files, is necessary to use the cluster. If you are unfamiliar with Linux, we recommend doing an introductory course, such as “Introduction to the Unix Shell” which can be found at www.ucl.ac.uk/isd/services/research-it/research-it-training
- Login nodes do not run cluster jobs, neither interactively nor in the background.
- File-servers present the storage (home directories, project space) to the login and compute nodes.
- A management node performs cluster management tasks and runs the scheduler (queue) that controls job throughput on the compute nodes.

HPC using Grid Engine

Anatomy of a Job Script

```
1. # This is an example of a job script.
2. #$ -S /bin/bash
3. #$ -l h_rt=0:10:0
4. #$ -l h_vmem=2G,tmem=2G
5. #$ -wd /home/users/dg/work
6. #$ -j y
7. #$ -N NCBIAllGenomes
8. # Commands to be executed go here...
9. /home/users/dg/SGEtest/md_single
```

HPC using Grid Engine

Anatomy of a Job Script

1. `#This is a simple example of a job script.` ← This is a comment (begins with a #).
2. `#$ -S /bin/bash` ← The first GE command. Note that the line begins with `#$`. It is NOT a comment, because of the `$` sign and requires the `#` sign, so do not remove it. The string following is a resource request. `-S` specifies the shell that the job will run under, followed by the path to Bash.
3. `#$ -l h_rt=0:10:0` ← Request the run-time for this job (in **hours:minutes:seconds**). *Note:* if you request a run-time that is insufficient for your job to complete, GE will kill your job when it has had its allotted time, with no warning (job no longer appears in the queue, premature output termination, no errors). Large over-provision of run-time in a request, may increase queuing time.

HPC using Grid Engine

Anatomy of a Job Script

4. `#$ -l h_vmem=2G,tmem=2G` ← This is a memory request and is required (otherwise the job will not run and simply stay in the queue). Note that there are two options in this request; `h_vmem` and `tmem`, and their values are given in gigabytes (`G`). Both these options must be given in a memory request and they should be the same value.

The technical reasons why memory is requested this way are beyond the scope of this course, but they allow GE to properly manage the memory of the job and keep track of memory usage on the compute node.

Requesting memory for a job is somewhat of an art. Requesting too much memory wastes resources and increases queuing time, whereas requesting too little causes the job process to swap, significantly increasing its run-time (which could mean it does not complete). It is important to be aware of an application's memory usage, so an appropriate memory request can be made.

Requesting memory rule of thumb: Benchmark your job! You can do this by requesting an interactive session on a compute node using the command `qssh -l h_vmem=nG,tmem=nG` where `n` is the amount of memory you wish to request, then running your application in the background. To do this, add an `&` symbol to the end of the command line. Once the application is running, use the `top` command to monitor the system resources, including memory, your application is using. It is only necessary to run the application in the background long enough to obtain a roughly consistent memory usage value. Use this value plus 0.5GB for the memory request in your job script.

HPC using Grid Engine

Anatomy of a Job Script

5. `#$ -wd /home/users/dg/work` ← This is the directory from where the job script was submitted.
6. `#$ -j y` ← This request means “join yes”. When a job is submitted, GE writes two status files to the home directory by default; a `.o<job-ID>` and a `.e<job-ID>`. The `.o` file contains *stdout* output which is generally anything that would normally be written to the screen if the application was run interactively. The `.e` file contains *stderr* output. It is normal, on job completion, for the `.e` file to be empty and as some error output may be written to the `.o` file anyway, to save generating a pair of files for each job, the `.e` file can be “joined” to the `.o` file. This simply means any output normally written to the `.e` file, will instead be written to the `.o` file and no `.e` file will be created.
7. `#$ -N NCBIAllGenomes` ← Name the job with the following string. If this request is not made, the job will take the name of the job script. The job name has an associated environment variable, `JOB_NAME`, which can be used to reference the job in a script (the name can also be used to hold other jobs from running until completion).

HPC using Grid Engine

Anatomy of a Job Script

8. `# Commands to be executed go here...` ← Another comment.

9. `/home/users/dg/SGEtest/md_single` ← The first normal shell command, usually the path to the application plus options, input and output directives, as would be given on the command line. As a job script is fundamentally a shell script, this could be additional shell commands, for example to pre-process data before the application is called. The shell commands in a job script should be written exactly as if the command was to be run interactively. It is a good idea to include the “`hostname`” and “`date`” commands as the first commands in the job script, so the node accepting the job and the start time can be determined. This is useful if the job errors in a way that suggests an issue with the node or the cluster.

HPC using Grid Engine

Managing Jobs

Once a job script has been created, it can be submitted to GE for processing. There are three principal commands for managing jobs in GE:

- **qsub** – submit a job script to the queue. The GE options in the job script can also be given on the qsub command line. There is an associated environment variable, JOB_ID, which can be used to reference the job:

```
dg@nexus:~/SGEtest$ qsub AllGenomeDownload.sh
Your job 25 ("NCBIAllGenomes") has been submitted
```

- **qdel** – delete a job from the queue. A job ID is required:

```
dg@nexus:~/SGEtest$ qdel 25<.${SGE_TASK_ID} (- array job)>
dg has deleted job 25
```
- **qstat** – check the status of a submitted job.

HPC using Grid Engine

Managing Jobs - qstat

```
dg@nexus:~/SGEtest$ qstat
```

job-ID	prior	name	user	state	submit/start at	queue	slots	ja-task-ID

25	0.55500	NCBIA11Gen	dg	r	04/23/2018 13:12:44	all.q@fry-611-48.local	1	

```
dg@nexus:~/SGEtest$
```

Fields to note:

- **job-ID** ← from qsub command.
- **name** ← from the **-N** option, if given in the job script. Otherwise, the job script name.
- **slot** ← number of cores the job is using (normally, no. of slots = no. of cores = no. of threads).
- **state** ← status of the job in the queue. There are four possible states: **qw**, **r**, **Eqw** and **s**.

HPC using Grid Engine

Managing Jobs – qstat states

- **qw** – “queue wait”. The normal state of a job that has just been submitted. The job is waiting for its requested resources to become available.
- **r** – “running”. The job is running!
- **Eqw** – “Error queue wait”. When a job is first submitted, GE evaluates the script to make sure there are no issues. If there are none, the job remains in qw state, waiting to run. If, however, an issue is found, the job goes into Eqw state. In this state, the job will remain in the queue, but it will not run until the error is corrected and the job resubmitted. The cause of the error can be found with the command “**qstat -f -j <job-ID>**” This gives a much more comprehensive description about the submitted job. Look for the string “**error reason**” in the output, to see what GE considers the issue to be.
- **s** – “Suspended”. If a job is in “s” state, it has been suspended. Jobs are suspended when they are causing issues on the cluster e.g. heavy I/O. During a suspension, no attempt should be made to restart the job, for example, by deleting and resubmitting it. Once the nature of the issue has been determined, the job will be resumed.

HPC using Grid Engine

Other Useful Job Script Options

- `#$ -o|e /alternative/directory/path` ← Use the `-o` and/or `-e` flag to redirect the `.o` and/or `.e` files to an alternative output directory (the default is the home directory, unless the `-cwd` or `-wd` flag was given).
- `#$ -cwd` ← This is the directory from where the job script was submitted.
- `#$ -V` ← Import the users `.bashrc` environment into the job.

HPC using Grid Engine

Array Jobs

- `#$ -t 1-10[:2]` ← Submits a so-called array job. An array job consists of a series of tasks. The tasks are scheduled independently and can run concurrently, very much like separate jobs. An array job can be accessed as a single unit or as individual tasks, by `qdel` for example.
- The option argument to `-t` specifies the number of array job tasks and the index number which will be associated with each task. The index numbers will be exported to the job tasks via the environment variable `SGE_TASK_ID`. The task id range specified may be a single number or a simple range. The `:2` is the task step size. If applied here, only tasks 1,3,5,7,9 would run. When absent, all tasks in the specified range are run. The task id is shown in the “`ja-task-ID`” output from `qstat`.
- `#$ -tc 5` ← Define the maximum number of concurrently running tasks for the array job. In this case, only the first 5 tasks would be run [or the first 5 tasks for the given step size].
- The STDOUT and STDERR of array job tasks will be written into different files with the default name: `<jobname>.e|o<job_id>.<task_id>`. In order to change this default, the `-e` and `-o` options can be used together with the environment variables `$JOB_NAME`, `$JOB_ID` and `$SGE_TASK_ID`.

HPC using Grid Engine

Array Jobs

- `$SGE_TASK_ID` can also be used to refer to lines in a file if, for example, there is a data-point on each line. This can be done in a few different ways, for example:
- `sed -n ${SGE_TASK_ID} '{p;q}' input.data`

The `-n` flag suppresses the 'normal' output of `sed` with `p`, which is to print the file given. The `q` quits `sed` as soon as the specified line is reached, which minimises unnecessary processing of the input file.

- `head -${SGE_TASK_ID} input.data | tail -1`

`head -[n]` prints the first N lines. `tail -[n]` prints the last N lines. In this case, `tail -1` is being run on a subset of the initial file (output of the `head` command), which has the Nth line as the final entry.

HPC using Grid Engine

Parallel Environments: SMP

- `#$ -pe smp 8` ← “[Request a] parallel environment, symmetric multi-processing [with] 8 [cores]”. Use this environment to run multi-threaded or OpenMP applications over multiple cores within a compute node. You must also include `#$ -R y` in your job script, to reserve the cores requested.
- In a parallel environment, the values of `tmem` and `h_vmem` are multiplied by the number of cores requested. This is because the scheduler interprets these values as being ‘per thread’ (where 1 thread = 1 core). So, in this example, if the memory request is 4GB, then the total memory resource the job requires is 32GB. In this environment, `tmem` and `h_vmem` should be adjusted to avoid significant queueing times.
- If the application has an option to execute using more than one thread or core, the number given should match that in the environment request. The variable NSLOTS can be used to achieve this. The application must support multi-threading.
- Please do not request a parallel environment of one core. Apart from the unnecessary scheduler overhead this creates, it does not make sense!

HPC using Grid Engine

Parallel Environment: MPI

- `#$ -pe mpi 8` ← “[Request a] parallel environment, message-passing interface [with] 8 [cores]”. Use this environment to run multi-threaded or OpenMPI applications across multiple compute nodes. The same factors that apply to SMP environments as described previously, also apply to MPI environments.
- MPI environments are particularly suited to workloads that require a substantial compute resources e.g. over 200 cores, so while they can be used run much smaller workloads, there is some additional complexity in running MPI-based applications and this is rarely necessary when the equivalent SMP environment is available.
- Please contact us if you are unfamiliar with running MPI applications and specifically need to do this.

HPC using Grid Engine

Accessing GPU Resources

- The GPU flag takes true as an argument: `#$ -l gpu=true`. This will schedule a job for a GPU node and request 1 GPU. **NOTE:** There is no `gpu=false` equivalent. If you no longer wish to use a GPU, simply remove the `gpu=true` flag. To request multiple GPUs, you must include the `-pe gpu` flag as follows: `#$ -pe gpu n` where `n` is the number of GPUs requested.
- Note that the `-pe gpu` flag augments the way the scheduler interprets `tmem` and `h_vmem`. These values will be multiplied by `n`, since the scheduler interprets them as being 'per thread' (or per GPU in this case - as for parallel environments). For example, a job requiring 4GB of memory and 2 GPUs would have a submission script containing: `#$ -l tmem=2G, #$ -l gpu=true, #$ -pe gpu 2, #$ -R y` and a job requiring 4GB of memory and a single GPU would include: `#$ -l tmem=4G, #$ -l gpu=true`.
- **NOTE:** `h_vmem` should be omitted for certain GPU jobs, specifically PyTorch and Tensorflow, as it causes memory allocation errors on the GPU.

HPC using Grid Engine

Interactive GPU Sessions

- The flags for requesting GPUs also apply when requesting interactive GPU sessions.
- You can combine the `-pe gpu n` directive with a `qrsh` request to test interactively across multiple GPUs: `qrsh -l tmem=4G,gpu=true,h_rt=0:60:0 -pe gpu 2`
- There is a fast test queue for interactive work with a short runtime (up to one hour) to enable you to test code quickly without having to wait: `qrsh -l tmem=14G,gpu=true,h_rt=0:30:0`
- Additionally, please only start up to two `qrsh` sessions per user (except those with group GPU nodes where you can request as many sessions as there are slots available).
- You no longer have to run the `CUDA_VISIBILITY.sh` script when running interactive/test sessions. The scheduler now handles all environment variables and the same applies with respect to manually setting the `CUDA_VISIBLE_DEVICES` variable. This variable controls which cards are visible and is set automatically. Please do not attempt to modify it from within your code, nor specify a GPU device index to use.

HPC using Grid Engine

Using Local Scratch Space

When performing large-scale job submissions, the I/O profile of the jobs must be taken into account, to avoid creating I/O bottle-necks that will slow-down job execution for everyone.

- Jobs performing significant I/O on input or output files, should be run from the local disk on the compute node. This can be requested in the job script using the `#$ -l tscratch=nG` option, where `n` is the amount of space requested in gigabytes.
- In the job script, include a command similar to:
`mkdir -p /scratch0/<username>/${JOB_ID} [.${SGE_TASK_ID} (array job)]`
- This is to prevent multiple jobs running on the same node from overwriting each other.
- Remember scratch space is for transient job data only and the data could be deleted without warning.

HPC using Grid Engine

Using Local Scratch Space

- When the application has executed, please delete the data from within the job script. To ensure this happens, incorporate a trap statement. For example, after the application command line:

```
function finish {  
    rm -rf /scratch0/<username>/${JOB_ID} [.${SGE_TASK_ID} (array job)]  
}  
  
trap finish EXIT ERR INT TERM
```

- This will run the finish function whenever the job exits, regardless of whether it finished successfully or not.

HPC using Grid Engine

Interactive Sessions on a Compute Node

- You can request an interactive session on the cluster using the `qrsh` command.
- The `qrsh` command requires a memory request using the `tmem` and `h_vmem` flags, just as in a job script e.g. “`qrsh -l tmem=2G,h_vmem=2G [-pe smp 2` (for multicore jobs adjusting the memory request accordingly)”
- As with job memory requests, the more memory you request for a `qrsh` session, the longer it may take for a session to be granted. If the requested resources are not available, the session will be refused.
- Unlike job scripts, a time request is not required; a `qrsh` session will last as long as needed.

HPC using Grid Engine

Applications

- Applications available to run on the cluster can be found in the directory `/share/apps`
- Within `/share/apps` there are directories for specific disciplines such as economics (`/share/apps/econ`) and biosciences (`/share/apps/genomics`). Some groups have their own directories, which can only be accessed by group members.
- Users can request applications to be installed centrally within `/share/apps` or are free to install applications in their home directories and project stores.
- Certain installed applications need an environment to be setup before they will run correctly. For those where this is the case, there are source files in `/share/apps/source_files`. Source the appropriate file (e.g. `source /share/apps/source_files/metaMix.source`) to set-up the application environment..
- There are additional example job scripts in `/share/apps/examples`

HPC using Grid Engine

Using Python

We have numerous versions of python3 (and the last version of python2) in `/share/apps`. To access any version of python, you must first “source” the source files for the python version you want to use, from `/share/apps/source_files/python`.

For example, if you want to use python-3.9.5-shared you must first run the command:
`source /share/apps/source_files/python/python-3.9.5.source`

This will add `/share/apps/python-3.9.5-shared/bin` to your `PATH` and `/share/apps/python-3.9.5-shared/lib` to `LD_LIBRARY_PATH`

HPC using Grid Engine

Using Python – Installing Packages

You can install packages for a particular python version without having to request us to install them centrally. To do this, first source the python version you want to install packages into (e.g. version 3.9.5). Then, determine if a package is installed by running the command

```
pip3 list | grep <package_name>
```

If the package is not found, it can be installed locally in your home directory, using the command:

```
python3 -m pip install --user <package_name>
```

This will install `<package_name>` into the `.local` directory in your home directory, under `3.9.5`. The package will be included with the centrally-installed site-packages of the python version you used to install it, when you run that version of python.

HPC using Grid Engine

Applying for Project Space

- Information about storage on the cluster can be found at: hpc.cs.ucl.ac.uk/data-storage. The storage request form is available here and this should be completed to apply for project space.
- This space is for processing research data on the cluster and is effectively mandatory.
- Home directories have strictly limited space and should not be used for data processing.
- You must confirm with your PI that you have permission to request the amount of project space you are requesting.

HPC using Grid Engine

File Management

Points to consider when performing large-scale job submissions:

- Use the `#$ -j y` flag, so GE only writes a `.o<JOB_ID>` file for each job/task.
- Running pipelines can typically result in a large number of output files being written to a common directory. This can slow the response of the file-system, which particularly impacts on data backups when they are run. Many of these files may be intermediate steps in the pipeline and can be deleted when the pipeline has completed, but otherwise, it is recommended that they be written to a tar archive. This reduces the file count, so relieving the impact on the file system.
- Note that the quota applied to project space on the CS cluster, includes a file count limit. Consequently, large numbers of small files can cause exceeded quota errors, even though relatively little disk space has been used.
- It is recommended that, where possible, large files such as those containing very large numbers of DNA sequences, should be processed in compressed form, if the application supports this.

HPC using Grid Engine

Help Requests

Please consider the following points when requesting help with job issues:

- Provide as much information as possible about the job that is experiencing the issue. This should include the path to the job script and job number, the application being run and any messages from GE or the application, before the issue was encountered.
- Include the “`hostname`” and “`date`” commands in the job script, so the node accepting the job and the start time can be determined. Sometimes, an issue with the node or the cluster can cause a job to error.
- There is more information about the CS cluster on the web-site: hpc.cs.ucl.ac.uk [username/password: “hpc”/“comic”] .

HPC using Grid Engine

Example Job Scripts

Single-Core Job

```
# Single-core job script example.
#$ -S /bin/bash
#$ -l h_rt=0:10:0
#$ -l h_vmem=2G,tmem=2G
#$ -wd /home/users/dg/work
#$ -j y
#$ -N md_single
# Commands to be executed follow.
hostname
date
/home/users/dg/SGEtest/md_single
```

HPC using Grid Engine

Example Job Scripts

Array Job

```
# Array job script example.
#$ -S /bin/bash
#$ -l h_rt=0:10:0
#$ -l h_vmem=2G,tmem=2G
#$ -wd /home/users/dg/work
#$ -j y
#$ -N md_single
#$ -t 1-100 # Array task range.
# SGE_TASK_ID is set to a number in the array task range.
# Commands to be executed follow.
hostname
date
/home/users/dg/SGEtest/md_single -log task.${SGE_TASK_ID}
```

HPC using Grid Engine

Example Job Scripts

Multi-Core SMP Job

```
# Multi-core SMP job script example.
#$ -S /bin/bash
#$ -l h_rt=0:10:0
#$ -l h_vmem=2G,tmem=2G # Initial memory values.
#$ -wd /home/users/dg/work
#$ -j y
#$ -N md_openmp
#$ -pe smp 16 # Core count multiplies initial memory values.
#$ -R y # Reserve (16) cores.
# Commands to be executed; NSLOTS sets threads to smp core count.
hostname
date
/home/users/dg/SGEtest/md_openmp -threads $NSLOTS
```

HPC using Grid Engine

Example Job Scripts

Multi-Core MPI Job

```
# Multi-core MPI job script example.
#!/bin/bash
#$ -S /bin/bash
#$ -l h_rt=02:00:00
#$ -l tmem=3.1G,h_vmem=3.1G # Initial memory values.
#$ -pe mpi 2 # Core count multiplies initial memory values.
#$ -R y # Reserve (2) cores.
#$ -wd /home/dgregory/metaMix # Path to working directory.
#$ -V # Include user environment.
# Commands to be executed.
hostname
date
export LD_LIBRARY_PATH=/share/apps/genomics/metaMix/openmpi-1.6.2/lib:$LD_LIBRARY_PATH
export PATH=/share/apps/genomics/metaMix/openmpi-1.6.2/bin:$PATH
# NSLOTS sets number of processors to mpi core count.
# mpirun starts the job and provides the communication interface.
mpirun --mca btl_tcp_if_include 128.41.97.0/21 -np $NSLOTS /share/apps/genomics/
metaMix/R-3.5.2/bin/R --slave CMD BATCH --no-save --no-restore Rsubmit.R step.out
```

HPC using Grid Engine

Example Job Scripts

GPU Job

```
# GPU job script example.
#$ -S /bin/bash
#$ -l h_rt=0:10:0
#$ -l tmem=2G # Initial memory value. Note no h_vmem.
#$ -wd /home/users/dg/work
#$ -j y
#$ -N md_gpu
#$ -l gpu=true
#$ -pe gpu 2 # GPU count multiplies initial memory value.
#$ -R y
# Commands to be executed follow.
hostname
date
/home/users/dg/SGEtest/md_gpu
```

HPC using Grid Engine

Example Job Scripts

Multi-Environment Job

```
# Multi-environment job script example.
#$ -S /bin/bash
#$ -l h_rt=240:00:0
#$ -l tmem=32G,h_vmem=32G
#$ -wd /home/users/dg/work
#$ -j y
#$ -N run_BEAST_L2
#$ -pe smp 16 # SMP environment.
#$ -R y
#$ -t 1-12 # Array tasks.
# Commands to be executed follow.
hostname
date
/share/apps/genomics/BEAST2-2.6.0/bin/beast -threads $NSLOTS 12_bmt_${SGE_TASK_ID}.xml
```