Grid Engine

Overview

When you run your jobs in Medusa, you must use Open Grid Scheduler/Grid Engine to submit the jobs.

The Grid Engine system recognizes the following four basic classes of jobs:

- **Batch Jobs** – Single segments of work. Typically, a batch job is only executed once.
- **Array Jobs** – Groups of similar work segments that can all be run in parallel but are completely independent of one another. All of the workload segments of an array job, known as tasks, are identical except for the data sets on which they operate.
- **Parallel Jobs** – Jobs composed of cooperating tasks that must all be executed at the same time, often with requirements about how the tasks are distributed across the resources.
- **Interactive Jobs** – Jobs that provide the submitting user with an interactive login to an available resource in the compute cluster. Interactive jobs allow users to execute work on the compute cluster that is not easily submitted as a batch job.

According to the class of your job, you should submit it properly as explained below.

Submitting Jobs (qsub)

To submit a job, you should use qsub command. In general, it looks like

```bash
$ qsub [resource requirements, options] -q <queue> <job_script>
```

- You should specify one of the queues to which you have been given to access. If you don't specify a queue, it will try to allocate to an available queue in the order of base.q, bigmem.q, and devel.q. For a serial job or a singjob job, you don't have to specify a queue if you don't mind where it would run. However, you should specify a queue for MPI jobs in general.
- If your job is expected to run more than 24 hours, you must specify the resource requirement, e.g., `-l h_rt=48:00:00` for 48 hours runtime. (See below.)
- If your job requires more than 3G memory per CPU core, you must specify the resource requirement, e.g., `-l h_vmem=4G` for 4G RAM. (See below.)

Here are some common options for the qsub, which can be used in the command line as the above or can be embedded with the line starting with `#` in the `<job_script>` file:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-cwd</td>
<td>executes the job from the current directory</td>
</tr>
<tr>
<td>-j y</td>
<td>merges the standard error with the standard output</td>
</tr>
<tr>
<td>-hold_jid &lt;job-IDs&gt;</td>
<td>waits for the specified &lt;job-IDs&gt; (yours) to be completed successfully (useful to resolve job dependencies)</td>
</tr>
<tr>
<td>-l &lt;resources&gt;</td>
<td>requests resources the job needs (See below for available resources)</td>
</tr>
</tbody>
</table>

Example of Submitting OpenMP Jobs

Example of Submitting MPI Jobs

Submitting Interactive Jobs (qrsh)

Monitoring Jobs (qstat)

Deleting Jobs (qdel)
-m <e|a|...> -M NetID@georgetown.edu sends email to the specified email address with "-M" at the end (e) of the job and/or when the job is aborted (a)...

-N <job_name> assigns the name of the job (defaults to <job_script>)

-pe <PE> NSLOTS specifies the Parallel Environment (PE) for OpenMP or MPI jobs (See below for available PEs)

-R y tries to reserve the required resources for a certain job (for example, a MPI job which requires a fair number of resources), keeping “small” jobs from filling in the queue ahead of the “big” job. This must be used with care.

-t <start-end[:step]> submits an Array job (See below)

Here are common resource requirements with -l <resources> option in Medusa:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>h_rt</td>
<td>Job's maximum runtime (wallclock). If your job runs longer than the default (24 hours), you must specify the requirement.</td>
<td>-l h_rt=24:00:00</td>
</tr>
<tr>
<td>h_vmem</td>
<td>Job's maximum virtual memory per CPU core. If your job requires more than the default (3G), you must specify the requirement.</td>
<td>-l h_vmem=3G</td>
</tr>
</tbody>
</table>

**Submitting Batch Jobs**

Here is a simple example for hello.c with Intel compiler:
#!/bin/bash
#
# Execute the job from the current directory.
#$ -cwd
#
# Merge the standard error with the standard output.
#$ -j y
#
# Send email at the end of the job. Please use your own email address.
#$ -m e -M chung@georgetown.edu
#
# Maximum runtime/wallclock. Please change it if your job requires more
# than the system default, 24 hours.
#$ -l h_rt=24:00:00
#
# Maximum virtual memory per CPU core. Please change it if your job
# requires more than the system default, 3G.
#$ -l h_vmem=3G
#
#
# Set up proper modules.
. /etc/profile
module load intel
module list

echo "Running on" `hostname`
#
# Run your batch/serial job.
./hello > hello.out

exit 0

$ module load intel
$ icc -o hello hello.c
$ qsub jb_batch.sh

Submitting Array Jobs

When you have the same executable needed to be run on many batch jobs with only different input data, for example, the Array Jobs may be more suitable, convenient, and controllable than submitting the many batch jobs. The option

```
-t <n-m>[s]
```

defines the task index range and each task can use the environment variable SGE_TASK_ID to retrieve its own task index number and to access input data sets designated for this task identifier, where n is the lowest index number, m the highest index number, and s the step size. Here is a simple example with pre-defined input data, data.in.1, data.in.2, ..., data.in.20.
#!/bin/bash

# Execute the job from the current directory.
#$ -cwd
#
# Merge the standard error with the standard output.
#$ -j y
#
# Send email at the end of the job. Please use your own email address.
#$ -m e-M chung@georgetown.edu
#
# Maximum runtime/wallclock. Please change it if your job requires more
# than the system default, 24 hours.
#$ -l h_rt=24:00:00
#
# Maximum virtual memory per CPU core. Please change it if your job
# requires more than the system default, 3G.
#$ -l h_vmem=3G
#
#
# Set up proper modules.
. /etc/profile
module load intel
module list

echo "Running on" `hostname`
# Run your array job.
if [ -f ./data.in.$SGE_TASK_ID ]
then
  ./hello < ./data.in.$SGE_TASK_ID > ./data.out.$SGE_TASK_ID
else
  echo "There is no input data.in.$SGE_TASK_ID!"
fi

exit 0

$ qsub -t 1-20 jb_array.sh

## Submitting Parallel Jobs

When you submit parallel jobs (OpenMP or MPI), you must specify the Parallel Environment (PE) with the option `-pe <PE> NSLOTS`, where `NSLOTS` is the number of processors to be used. The following are available PEs in Medusa:

<table>
<thead>
<tr>
<th>PE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>smp</td>
<td>All slots are allocated on the same node, so <code>NSLOTS</code> should be less than or equal to the available number of CPU cores on the node.</td>
</tr>
</tbody>
</table>
Slots are filled up on one node, move to another, and continue until range filled.

Round-robin allocation over all suitable nodes until range filled.

1 slot per node

Example of Submitting OpenMP Jobs

OpenMP (Open Multi-Processing) applications must use `smp` PE with the environment variable OMP_NUM_THREADS. Here is an example:

```
#!/bin/bash
#
# Execute the job from the current directory.
#$ -cwd
#
# Merge the standard error with the standard output.
#$ -j y
#
# Send email at the end of the job. Please use your own email address.
#$ -m e -M chung@georgetown.edu
#
# Maximum runtime/wallclock. Please change it if your job requires more than the system default, 24 hours.
#$ -l h_rt=24:00:00
#
# Maximum virtual memory per CPU core. Please change it if your job requires more than the system default, 3G. (Total Memory = h_vmem * $NSLOTS)
#$ -l h_vmem=3G
#
# Set up proper modules.
. /etc/profile
module load intel
module list

# The following must be defined for OpenMP job.
export OMP_NUM_THREADS=$NSLOTS

echo "Got $NSLOTS slots running on" `hostname`
# Run your OpenMP job
./hello_omp > hello_omp.out

exit 0
```

$ module load intel
$ icc -openmp -o hello_omp hello_omp.c
$ qsub -pe smp 8 jb_openmp.sh
Example of Submitting MPI Jobs

MPI applications must use one of the above available PEs, *smp*, *mpi* or *mpi_r*. There are two MPI implementations available in Medusa, *Intel MPI* and *Open MPI*. Both MPI implementations can share the same PEs and job scripts. Just choose one with the *modules*. Please use *mpirun* instead of *mpiexec*. Here is an example:

```bash
#!/bin/bash
#
# Execute the job from the current directory.
#$ -cwd
#
# Merge the standard error with the standard output.
#$ -j y
#
# Send email at the end of the job. Please use your own email address.
#$ -m e -M chung@georgetown.edu
#
# Maximum runtime/wallclock. Please change it if your job requires more
# than the system default, 24 hours.
#$ -l h_rt=24:00:00
#
# Maximum virtual memory per CPU core. Please change it if your job
# requires more than the system default, 3G. (Total Memory = h_vmem *
#$ -l h_vmem=3G
#
# Set up proper modules.
. /etc/profile
module load intel-mpi    # intel-mpi or open-mpi
module list

echo "Got $NSLOTS slots."
# Run your MPI job.
mpirun -n $NSLOTS ./hello_mpi > hello_mpi.out

exit 0
```

For MPI jobs, the email notifications from the Grid Engine may have incorrect information for some contents such as CPU/user/system time. The correct information can be obtained by the following command (unit in seconds):

```
$ qacct -j <job_id>
```

Please refer to *Intel Parallel Studio XE* and *Open MPI* for more information.
Submitting Interactive Jobs *(qrsh)*

If you need to run jobs interactively on compute nodes, you should use `qrsh` command instead of `qsub`. For example,

```
$ qrsh [resource requirements] -q <queue> [command]
```

If no command is specified, a remote `qlogin` session is started on a compute node.

Monitoring Jobs *(qstat)*

To monitor currently submitted jobs, use the `qstat` command:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>qstat</code></td>
<td>shows your jobs</td>
</tr>
<tr>
<td><code>qstat -u ***</code></td>
<td>shows all users’ jobs</td>
</tr>
<tr>
<td><code>qstat -f</code></td>
<td>specifies a &quot;full&quot; format display of information on all queues</td>
</tr>
<tr>
<td><code>qstat -j &lt;job_ids or job_names&gt;</code></td>
<td>prints various information of the jobs</td>
</tr>
<tr>
<td><code>qstat -g c</code></td>
<td>displays a cluster queue summary</td>
</tr>
<tr>
<td><code>qstat -F h_vmem</code></td>
<td>shows available h_vmem resources on all queues</td>
</tr>
</tbody>
</table>

Deleting Jobs *(qdel)*

To delete your submitted job,

```
$ qdel <job_id>
```

To delete specified tasks of array job,

```
$ qdel <job_id> -t <start-end[:step]>
```