Running Jobs with SLURM (Puma)

Overview

SLURM
The new HPC system, Puma, uses SLURM as a job scheduler rather than PBS Pro. SLURM has several advantages:

- It provides more robust support for a larger number of jobs in queue.
- It is used by national HPC groups (XSEDE and TACC) making it easier for users to scale out to those systems.
- It has more sustainable support.

Allocations and Job Partitions (Queues)
Using Puma with SLURM is similar to using ElGato and Ocelote with PBS. Users will still receive a monthly allocation of cpu hours associated with their PI's group which will be deducted when they run their jobs in standard. Users will also still be able to use windfall to run jobs without consuming their monthly allocations. As on Ocelote and Puma, jobs run using windfall will still be subject to preemption when resources are requested by higher-priority jobs.

To request a specific partition (standard, windfall, or high_priority), see Job Partition Requests below.

Modules and Software
The process of finding, loading, and using software as modules will not change on the new system. Users will still be able to utilize the standard commands described in the Software section in our User Guide. However, in a departure from our previous systems, modules will not be available to load and utilize on the login nodes. To load, use, and test software for job submissions, users will need to request an interactive session. Interactive sessions may be requested by simply using the command “interactive”.

PBS SLURM Rosetta Stone
In general, SLURM can translate and execute scripts written for PBS. This means that if you submit a PBS script written for Ocelote or ElGato on Puma (with the necessary resource request modifications), your script will likely run. However, there are a few caveats that should be noted:

- You will need to submit your job with the new SLURM commands, e.g. sbatch instead of qsub
- There may be some PBS directives that do not directly translate to SLURM which cannot be interpreted
- The environment variables specific to PBS and SLURM are different. If your job relies on these, you will need to update them. Common examples are PBS_O_WORKDIR and PBS_ARRAY_INDEX

To help with the transition to SLURM, we've also installed software that converts some basic PBS Pro commands into SLURM commands automatically called pbs2slurm.
To get acquainted with the new scheduling system, refer to the following list of common PBS commands, directives, and environment variables and their SLURM counterparts.

<table>
<thead>
<tr>
<th>PBS</th>
<th>SLURM</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Job Management</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>qsub &lt;options&gt;</code></td>
<td><code>sbatch &lt;options&gt;</code></td>
<td>Batch submission of jobs to run without user input</td>
</tr>
<tr>
<td><code>qsub -I &lt;options&gt;</code></td>
<td><code>srun &lt;options&gt; -pty bash -i</code></td>
<td>Request an interactive job</td>
</tr>
<tr>
<td></td>
<td><code>salloc &lt;options&gt;</code></td>
<td>Request an interactive job</td>
</tr>
<tr>
<td><code>qstat</code></td>
<td><code>squeue</code></td>
<td>Show all jobs</td>
</tr>
<tr>
<td><code>qstat &lt;jobid&gt;</code></td>
<td><code>squeue --job &lt;jobid&gt;</code></td>
<td>Check status of a specific job</td>
</tr>
<tr>
<td><code>qstat -u &lt;netid&gt;</code></td>
<td><code>squeue -u &lt;netid&gt;</code></td>
<td>Check status of jobs specific to user</td>
</tr>
<tr>
<td><code>tracejob &lt;jobid&gt;</code></td>
<td><code>sacct -j &lt;jobid&gt;</code></td>
<td>Check history of a completed job</td>
</tr>
<tr>
<td><code>qdel &lt;jobid&gt;</code></td>
<td><code>scancel &lt;jobid&gt;</code></td>
<td>Delete a specific job</td>
</tr>
<tr>
<td><code>qdel -u &lt;netid&gt;</code></td>
<td><code>scancel -u &lt;netid&gt;</code></td>
<td>Delete all user jobs</td>
</tr>
<tr>
<td><code>qstat -Q</code></td>
<td><code>sinfo</code></td>
<td>View information about nodes and queues</td>
</tr>
<tr>
<td><code>qhold &lt;jobid&gt;</code></td>
<td><code>scontrol hold &lt;jobid&gt;</code></td>
<td>Places a hold on a job to prevent it from being executed</td>
</tr>
<tr>
<td><code>qrls &lt;jobid&gt;</code></td>
<td><code>scontrol release &lt;jobid&gt;</code></td>
<td>Releases a hold placed on a job allowing it to be executed</td>
</tr>
</tbody>
</table>

**Job Directives**

```
#PBS -W group_list=group_name
#SBATCH --account=group_name
```
Specify group name where hours are charged

```
#PBS -q standard
#SBATCH --partition=standard
```
Set job queue

```
#PBS -l walltime=HH:MM:SS
#SBATCH --time HH:MM:SS
```
Set job walltime

```
#PBS -l select=<N>
#SBATCH --nodes=<N>
```
Select N nodes

```
#PBS -l ncpus=<N>
#SBATCH --cpus-per-task=<M>
```
PBS: Select N cpus
SLURM: Each task is assume to require one cpu. Optionally, you may include `cpus-per-task` if more are required. Requests `NxM` cpus

```
#PBS -l mem=<N>gb
#SBATCH --mem=<N>gb
```
Select N gb of memory

```
#PBS -l pvmem=<N>gb
#SBATCH --mem-per-cpu=<N>gb
```
Select N gb of memory per cpu

```
#PBS J N-M
#SBATCH --array=N-M
```
Array job submissions where N and M are integers

```
#PBS -l np100s=1
#SBATCH --gres=gpu:1
```
Optional: Request a GPU

```
#PBS -N JobName
#SBATCH --job-name=JobName
```
Optional: Set job name

```
#PBS -j oe
(default)
#SBATCH -e <job_name>%j.
err
#SBATCH -o <job_name>%j.
out
```
Optional: Combine stdout and stderr

```
(default)
#SBATCH -o filename
#SBATCH -e filename
```
Optional: Standard output filename

```
#SBATCH -o filename
#SBATCH -e filename
```
Optional: Error filename

```
#SBATCH --open-mode=append
```
Optional: Combine all output into single file. Note: If this is selected, each job run will append to that filename, including preexisting files with that name

Environment Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PBS_O_WORKDIR</td>
<td>Job submission directory</td>
</tr>
<tr>
<td>$SLURM_SUBMIT_DIR</td>
<td></td>
</tr>
<tr>
<td>$PBS_JOBID</td>
<td>Job ID</td>
</tr>
<tr>
<td>$PBS_JOBNAME</td>
<td>Job name</td>
</tr>
<tr>
<td>$PBS_ARRAY_INDEX</td>
<td>Index to differentiate tasks in an array</td>
</tr>
<tr>
<td>$PBS_NODEFILE</td>
<td>List of nodes allocated to current job</td>
</tr>
<tr>
<td>$PBS_O_HOST</td>
<td>Hostname where job was submitted</td>
</tr>
<tr>
<td>$SLURM_SUBMIT_HOST</td>
<td></td>
</tr>
</tbody>
</table>

Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue</td>
<td>Partition</td>
</tr>
<tr>
<td>Group List</td>
<td>Association</td>
</tr>
<tr>
<td>PI</td>
<td>Account</td>
</tr>
</tbody>
</table>

Job Partition Requests

SLURM partition requests are slightly different from PBS. Use the following table as a guide for how to use the partition that is relevant to you:

<table>
<thead>
<tr>
<th>Partition</th>
<th>SLURM</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard</td>
<td>#SBATCH --account=&lt;PI GROUP&gt; #SBATCH --partition=standard</td>
<td>Consumes your group's standard allocation.</td>
</tr>
<tr>
<td>windfall</td>
<td>#SBATCH --partition=windfall</td>
<td>Does not consume your group's standard allocation. Jobs may be interrupted and restarted by higher-priority jobs. The --account flag needs to be omitted or an error will occur.</td>
</tr>
<tr>
<td>high_priority</td>
<td>#SBATCH --partition=&lt;PI GROUP&gt; #SBATCH --partition=standard #SBATCH --qos=user_qos_&lt;$NETID&gt;</td>
<td>Available for groups who have purchased compute resources. The partition flag is left as standard and requires the additional qos flag. Replace &lt;NETID&gt; with your own UArizona NetID.</td>
</tr>
</tbody>
</table>

SLURM Output Filename Patterns

Unlike PBS, SLURM offers ways to make your job's output filenames more customizable through the use of character replacements. A table is provided below as a guide with some examples. Variables may be used or combined as desired. Note: character replacements may also be used with other SBATCH directives such as error filename, input filename, and job name.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
<th>Example Slurm Directive(s)</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>%A</td>
<td>A job array's main job ID</td>
<td>#SBATCH --array=1-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>#SBATCH -o=%A.out</td>
<td>12345.out</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#SBATCH --open-mode=append</td>
<td></td>
</tr>
<tr>
<td>%a</td>
<td>A job array's index number</td>
<td>#SBATCH --array=1-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>#SBATCH -o=%A_%a.out</td>
<td></td>
</tr>
<tr>
<td>%J</td>
<td>Job ID plus stepid</td>
<td>#SBATCH -o=%J.out</td>
<td>12345.out</td>
</tr>
<tr>
<td>%j</td>
<td>Job ID</td>
<td>#SBATCH -o=%j.out</td>
<td>12345.out</td>
</tr>
<tr>
<td>%N</td>
<td>Hostname of the first compute node allocated to the job</td>
<td>#SBATCH -o=%N.out</td>
<td>r1u11n1.out</td>
</tr>
<tr>
<td>%u</td>
<td>Username</td>
<td>#SBATCH -o=%u.out</td>
<td>netid.out</td>
</tr>
<tr>
<td>%x</td>
<td>Jobname</td>
<td>#SBATCH --job-name=JobName</td>
<td>JobName.out</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#SBATCH -o=%x.out</td>
<td></td>
</tr>
</tbody>
</table>

**Job Examples**

**Single serial job submission**

**PBS Script**

```bash
#!/bin/bash
#PBS -N Sample_PBS_Job
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -l walltime=00:01:00
#PBS -q windfall
#PBS -W group_list=<group_name>
cd $PBS_O_WORKDIR
pwd; hostname; date
module load python
python --version
```

**SLURM Script**

```bash
#!/bin/bash
#SBATCH --job-name=Sample_Slurm_Job
#SBATCH --ntasks=1
#SBATCH --mem=1gb
#SBATCH --time=00:01:00
#SBATCH --partition=windfall
#SBATCH --account=<group_name>

# SLURM Inherits your environment. cd
#$SLURM_SUBMIT_DIR not needed
pwd; hostname; date

module load python
python --version
```

**Array Submission**
IMPORTANT:
When submitting jobs with named output files (i.e. with the line `#SBATCH -o=Job.out`) as arrays, SLURM will write every array element to that filename leaving you with only the output of the last completed job in the array. Use one of the following SLURM directives in your script to prevent this behavior:

1. Differentiates output files using array indices. Similar to PBS default. See SLURM Output Filename Patterns above for more information.
   
   ```bash
   #SBATCH --output=Job-%a.out
   ```

2. Appends the output from all tasks in an array to the same output file. Warning: if a file exists with that name prior to running your job, the output will be appended to that file.
   
   ```bash
   #SBATCH --open-mode=append
   ```

### PBS Script

```bash
#!/bin/bash
#PBS -N Sample_PBS_Job
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -l walltime=00:01:00
#PBS -q windfall
#PBS -W group_list=<group_name>
#PBS -J 1-5

cd $PBS_O_WORKDIR
pwd; hostname; date

echo "./sample_command input_file_${PBS_ARRAY_INDEX}.in"
```

### SLURM Script

```bash
#!/bin/bash
#SBATCH --output=Sample_SLURM_Job-%a.out
#SBATCH --ntasks=1
#SBATCH --mem=1gb
#SBATCH --time=00:01:00
#SBATCH --partition=windfall
#SBATCH --account=<group_name>
#SBATCH --array 1-5

# SLURM Inherits your environment. cd
$SLURM_SUBMIT_DIR not needed
pwd; hostname; date

echo "./sample_command input_file_${SLURM_ARRAY_TASK_ID}.in"
```

### MPI Example

```bash
#!/bin/bash
```
#!/bin/bash
#PBS -N Sample_MPI_Job
#PBS -l select=1:ncpus=16:mem=16gb
#PBS -l walltime=00:10:00
#PBS -W group_list=<group_name>
#PBS -q standard

cd $PBS_O_WORKDIR
pwd; hostname; date
module swap openmpi mpich
/usr/bin/time -o mpit_prog.timing mpirun -np 16 a.out

---

SLURM Script

#!/bin/bash
#SBATCH --job-name=Sample_MPI_Job
#SBATCH --ntasks=16
#SBATCH --ntasks-per-node=16
#SBATCH --nodes=1
#SBATCH --mem-per-cpu=1gb
#SBATCH --time=00:10:00
#SBATCH --account=<group_name>
#SBATCH --partition=standard
#SBATCH --output=Sample_MPI_Job_%A.out
#SBATCH --error=Sample_MPI_Job_%A.err

# SLURM Inherits your environment. cd
$SLURM_SUBMIT_DIR not needed
pwd; hostname; date
module swap openmpi3 mpich/3.3.1
/usr/bin/time -o mpit_prog.timing mpirun -np 16 a.out

---

rosetta_min.pdf