Running Jobs with SLURM

Overview

SLURM

All three clusters, Puma, Ocelote, and ElGato, use 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 SLURM is similar to using PBS. Users will still receive a monthly allocation of CPU hours associated with their PI's group that will be deducted when they run their jobs in standard. Users will also be able to use windfall to run jobs without consuming their monthly allocations. Jobs run using windfall will 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.

Resources per node

This table shows the resources available in each node that are used in the Slurm script:

<table>
<thead>
<tr>
<th>Compute Resources</th>
<th>Example Resource Requests</th>
</tr>
</thead>
</table>

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 are not 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" (see section below).

Interactive command

When you are on a login node, you can request an interactive session on a compute node. This is useful for checking available modules, testing submission scripts, compiling software, and running programs directly from the command line. To get an interactive session, we have a built in command that will allow you to quickly and easily do so by simply entering:
Submitting this actually runs the following:

```
salloc --job-name=interactive --mem-per-cpu=4GB --nodes=1 --ntasks=1 --time=01:00:00 --account=windfall --partition=windfall
```

If you find that this session is insufficient, `interactive` has built-in customization flags. For example, if you want to get a session faster, add your PI's account name to use the standard partition:

```
interactive -a account_name
```

Are you using X11 forwarding?

```
interactive -a account_name -x
```

Full usage:

```
interactive [-x] [-N nodes] [-n ncpus per node] [-Q optional qos] [-t hh:mm:ss] [-a account to charge]
```

Any time you submit an `interactive` command, it will always print the full `salloc` being executed for verification and copying/editing/pasting.

**GPU Jobs**

To request a GPU, you will include the resource name using the `--gres` SLURM directive. For example, if you wanted to request an interactive session with one GPU, you could run:

```
salloc --job-name=interactive --mem-per-cpu=4GB --nodes=1 --ntasks=1 --time=01:00:00 --account=windfall --gres=gpu:1
```

In a batch script, you would include the number of GPUs as an `SBATCH` directive. For example:

```
#SBATCH --gres=gpu:1
```

In both cases above, the jobs are requesting 1 GPU. This number can be increased up to 4 on Puma and 2 on Ocelote depending on the number of GPUs you need for your workflow.

Note that on Ocelote there are two models of GPU's, K80 and P100. If you don't specify which one, you could get either. So for ocelote, we recommend:

```
#SBATCH --gres=gpu:pascal:1
```

**High Memory Nodes**

Puma has two high memory nodes available with 3TB of RAM each. These nodes have a ratio of 32GB of RAM per CPU, so a job requesting N CPUs would be allocated N*32GB of RAM. To request one, you may either explicitly set `--mem-per-cpu=32gb`, or `--constraint=hi_mem` in your job script. For example, the following directives:

```
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=5
#SBATCH --mem-per-cpu=32gb
```

would run a job on one of the high memory nodes with 160GB of RAM. The following would request identical resources:
#SBATCH --ntasks=1  
#SBATCH --cpus-per-task=5  
#SBATCH --constraint=hi_mem

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 command, 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 get acquainted with the new scheduling system, refer to the following list of common PBS commands, directives, and environment variables and their SLURM counterparts. For a PDF version, click here.

<table>
<thead>
<tr>
<th>PBS</th>
<th>SLURM</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<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>salloc &lt;options&gt;</code></td>
<td>Request an interactive job</td>
</tr>
<tr>
<td>N/A</td>
<td><code>srun &lt;options&gt;</code></td>
<td>Submit a job for realtime execution. Can also be used to submit an interactive session</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

<table>
<thead>
<tr>
<th>PBS</th>
<th>SLURM</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#PBS -W group_list=group_name</code></td>
<td><code>#SBATCH --account=group_name</code></td>
<td>Specify group name where hours are charged</td>
</tr>
<tr>
<td><code>#PBS -q standard</code></td>
<td><code>#SBATCH --partition=standard</code></td>
<td>Set job queue</td>
</tr>
<tr>
<td><code>#PBS -l walltime=HH:MM:SS</code></td>
<td><code>#SBATCH --time HH:MM:SS</code></td>
<td>Set job walltime</td>
</tr>
<tr>
<td><code>#PBS -l select=&lt;N&gt;</code></td>
<td><code>#SBATCH --nodes=&lt;N&gt;</code></td>
<td>Select N nodes</td>
</tr>
<tr>
<td><code>#PBS -l ncpus=&lt;N&gt;</code></td>
<td><code>#SBATCH --ntasks=&lt;N&gt;</code></td>
<td>PBS: Select N cpus</td>
</tr>
<tr>
<td></td>
<td><code>#SBATCH --cpus-per-task=&lt;M&gt;</code></td>
<td>SLURM: Each task is assume to require one cpu. Optionally, you may include cpus-per-task if more are required. Requests NxM cpus</td>
</tr>
<tr>
<td><code>#PBS -l mem=&lt;N&gt;gb</code></td>
<td><code>#SBATCH --mem=&lt;N&gt;gb</code></td>
<td>Note: Puma has 94 cpus available on each node</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Select N gb of memory per node</td>
</tr>
</tbody>
</table>
#PBS -l pcmem=<N>gb
#SBATCH --mem-per-cpu=<N>gb
Select N gb of memory per cpu
Note: Puma defaults to 5GB 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
#SBATCH --job-rename=JobName
Optional: Combine stdout and error
(default)
#SBATCH -e <job_name>-%j.err
#SBATCH -o <job_name>-%j.out
Optional: Separate stdout and stderr
(SLURM: %j is a stand-in for $SLURM_JOB_ID)
#PBS -o filename
#SBATCH -o filename
Optional: Standard output filename
#PBS -e filename
#SBATCH -e filename
Optional: Error filename
N/A
#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
#PBS -v var=<value>
#SBATCH --export=var
Optional: Export single environment variable var to job
#PBS --export=none
Optional: Do not export working environment to job
#PBS -m be
#SBATCH --mail-type=BEGIN|END|FAIL|ALL
Optional: Request email notifications
Beware of mail bombing yourself
#PBS -M <netid>@email.arizona.edu
#SBATCH --mail-user=<netid>@email.arizona.edu
Optional: email address used for notifications
#PBS -l place=excl
#SBATCH --exclusive
Optional: Request exclusive access to node

Environment Variables

$PBS_O_WORKDIR $SLURM_SUBMIT_DIR Job submission directory
$PBS_JOBID $SLURM_JOB_ID Job ID
$PBS_JOBNAME $SLURM_JOB_NAME Job name
$PBS_ARRAY_INDEX $SLURM_ARRAY_TASK_ID Index to differentiate tasks in an array
$PBS_HOST $SLURM_SUBMIT_HOST Hostname where job was submitted
$PBS_NODEFILE $SLURM_JOB_NODELIST List of nodes allocated to current job

Terminology

Queue Partition
Group List Association
PI Account

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>
</table>

<p>| PBS_O_WORKDIR | SLURM_SUBMIT_DIR | Job submission directory |
| PBS_JOBID | SLURM_JOB_ID | Job ID |
| PBS_JOBNAME | SLURM_JOB_NAME | Job name |
| PBS_ARRAY_INDEX | SLURM_ARRAY_TASK_ID | Index to differentiate tasks in an array |
| PBS_HOST | SLURM_SUBMIT_HOST | Hostname where job was submitted |
| PBS_NODEFILE | SLURM_JOB_NODELIST | List of nodes allocated to current job |</p>
<table>
<thead>
<tr>
<th>Account Type</th>
<th>SBATCH Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard</td>
<td><code>#SBATCH --account=&lt;PI GROUP&gt;</code>&lt;br&gt;<code>#SBATCH --partition=standard</code></td>
<td>Consumes your group's standard allocation.</td>
</tr>
<tr>
<td>windfall</td>
<td><code>#SBATCH --partition=windfall</code></td>
<td>Does not consume your group's standard allocation. Jobs may be interrupted and restarted by higher-priority jobs. The <code>--account</code> flag needs to be omitted or an error will occur.</td>
</tr>
<tr>
<td>high_priority</td>
<td><code>#SBATCH --account=&lt;PI GROUP&gt;</code>&lt;br&gt;<code>#SBATCH --partition=standard</code>&lt;br&gt;<code>#SBATCH --qos=user_qos_&lt;PI GROUP&gt;</code></td>
<td>Available for groups who have purchased compute resources. The partition flag is left as standard and requires the additional <code>--qos</code> flag. Replace <code>&lt;PI GROUP&gt;</code> with your group's name.</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><code>#SBATCH --array=1-2</code>&lt;br&gt;<code>#SBATCH --o %A.out</code>&lt;br&gt;<code>#SBATCH --open-mode=append</code></td>
<td>12345.out</td>
</tr>
<tr>
<td>%a</td>
<td>A job array's index number</td>
<td><code>#SBATCH --array=1-2</code>&lt;br&gt;<code>#SBATCH --o %A_%a.out</code></td>
<td>12345_1.out&lt;br&gt;12345_2.out</td>
</tr>
<tr>
<td>%J</td>
<td>Job ID plus stepid</td>
<td><code>#SBATCH --o %J.out</code>&lt;br&gt;<code>#SBATCH --o %A_%a.out</code></td>
<td>12345.out</td>
</tr>
<tr>
<td>%j</td>
<td>Job ID</td>
<td><code>#SBATCH --o %j.out</code>&lt;br&gt;<code>#SBATCH --o %A_%a.out</code></td>
<td>12345.out</td>
</tr>
<tr>
<td>%N</td>
<td>Hostname of the first compute node allocated to the job</td>
<td><code>#SBATCH --o %N.out</code>&lt;br&gt;<code>#SBATCH --o %A_%a.out</code></td>
<td>r1u11n1.out&lt;br&gt;r1u11n1.out</td>
</tr>
<tr>
<td>%u</td>
<td>Username</td>
<td><code>#SBATCH --o %u.out</code>&lt;br&gt;<code>#SBATCH --o %A_%a.out</code></td>
<td>netid.out&lt;br&gt;netid.out</td>
</tr>
<tr>
<td>%x</td>
<td>Jobname</td>
<td><code>#SBATCH --o %x.out</code>&lt;br&gt;<code>#SBATCH --job-name=JobName</code>&lt;br&gt;<code>#SBATCH --o %A_%a.out</code></td>
<td>JobName.out&lt;br&gt;JobName.out</td>
</tr>
</tbody>
</table>

---

**Job Examples**

- For more examples, see our Github page!

---

**Single serial job submission**

```bash
PBS Script
```
#!/bin/bash

#PBS -N Sample_PBS_Job
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -l walltime=00:01:00
#PBS -q standard
#PBS -W group_list=<group_name>

cd $PBS_O_WORKDIR
pwd; hostname; date

module load python
python --version

---

SLURM Script

#!/bin/bash

#SBATCH --job-name=Sample_Slurm_Job
#SBATCH --ntasks=1
#SBATCH --nodes=1
#SBATCH --mem=1gb
#SBATCH --time=00:01:00
#SBATCH --partition=standard
#SBATCH --account=<group_name>

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

module load python/3.6
python3 --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.

   #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

   #SBATCH --open-mode=append

---

PBS Script
MPI Example

For openmpi the important variables are set by default, so you do not need to include them in your scripts.

Default OpenMPI variables

- export SBATCH_GET_USER_ENV=1
- export OMP1_MCA_btl_openib_cpc_include=rdmacm
- export OMP1_MCA_btl_openib_if_include=bnxt_rel
- export OMP1_MCA_btl_openib_rroce_enable=1
- export OMP1_MCA_btl_vader_self_openib
- export OMP1_MCA_oob_tcp_if_include=eth1

For Intel MPI, these variables are set for you:

Default Intel MPI variables

- export I_MPI_FABRICS=shm:ofi
- export FI_PROVIDER=verbs
- export FI_VERBS_IFACE=eth1
#!/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 load openmpi
/usr/bin/time -o mpitprog.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 load openmpi3
/usr/bin/time -o mpitprog.timing mpirun -np 16 a.out