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".

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:

```
interactive
```

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 run a job on a GPU you request the resource name that is assigned to the GPU. So in this example interactive job:

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

The resource is gpu and the quantity is 1.

**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 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><code>qsub</code> &lt;options&gt;</td>
<td><code>sbatch</code> &lt;options&gt;</td>
<td>Batch submission of jobs to run without user input</td>
</tr>
<tr>
<td><code>qsub</code> -I &lt;options&gt;</td>
<td><code>srun</code> &lt;options&gt; --pty bash -i</td>
<td>Request an interactive job</td>
</tr>
<tr>
<td><code>salloc</code> &lt;options&gt;</td>
<td><code>srun</code> &lt;options&gt;</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</code> --job &lt;jobid&gt;</td>
<td>Check status of a specific job</td>
</tr>
<tr>
<td><code>qstat -u &lt;netid&gt;</code></td>
<td><code>squeue</code> -u &lt;netid&gt;</td>
<td>Check status of jobs specific to user</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><code>tracejob &lt;jobid&gt;</code></td>
<td>Check history of a completed job</td>
<td></td>
</tr>
<tr>
<td><code>qdel &lt;jobid&gt;</code></td>
<td>Delete a specific job</td>
<td></td>
</tr>
<tr>
<td><code>qdel -u &lt;netid&gt;</code></td>
<td>Delete all user jobs</td>
<td></td>
</tr>
<tr>
<td><code>qstat -Q</code></td>
<td>View information about nodes and queues</td>
<td></td>
</tr>
<tr>
<td><code>qhold &lt;jobid&gt;</code></td>
<td>Places a hold on a job to prevent it from being executed</td>
<td></td>
</tr>
<tr>
<td><code>qrls &lt;jobid&gt;</code></td>
<td>Releases a hold placed on a job allowing it to be executed</td>
<td></td>
</tr>
</tbody>
</table>

### Job Directives

<table>
<thead>
<tr>
<th>PBS</th>
<th>SBATCH</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-W group_list=group_name</code></td>
<td><code>--account=group_name</code></td>
<td>Specify group name where hours are charged</td>
</tr>
<tr>
<td><code>-q standard</code></td>
<td><code>--partition=standard</code></td>
<td>Set job queue</td>
</tr>
<tr>
<td><code>-l walltime=HH:MM:SS</code></td>
<td><code>--time HH:MM:SS</code></td>
<td>Set job walltime</td>
</tr>
<tr>
<td><code>-l select=&lt;N&gt;</code></td>
<td><code>--nodes=&lt;N&gt;</code></td>
<td>Select N nodes</td>
</tr>
<tr>
<td><code>-l ncpus=&lt;N&gt;</code></td>
<td><code>--ntasks=&lt;N&gt;</code></td>
<td>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. Note: Puma has 94 cpus available on each node</td>
</tr>
<tr>
<td><code>-l mem=&lt;N&gt;gb</code></td>
<td><code>--mem=&lt;N&gt;gb</code></td>
<td>Select N gb of memory</td>
</tr>
<tr>
<td><code>-p mem=&lt;N&gt;gb</code></td>
<td><code>--mem-per-cpu=&lt;N&gt;gb</code></td>
<td>Select N gb of memory per cpu. Note: Puma defaults to 5GB per cpu</td>
</tr>
<tr>
<td><code>-l J N-M</code></td>
<td><code>--array=N-M</code></td>
<td>Array job submissions where N and M are integers</td>
</tr>
<tr>
<td><code>-l np100s=1</code></td>
<td><code>--gres=gpu:1</code></td>
<td>Optional: Request a GPU</td>
</tr>
<tr>
<td><code>-N JobName</code></td>
<td><code>--job-name=JobName</code></td>
<td>Optional: Set job name</td>
</tr>
<tr>
<td><code>-j oe</code></td>
<td>(default)</td>
<td>Optional: Combine stdout and error</td>
</tr>
<tr>
<td><code>-o filename</code></td>
<td><code>--o filename</code></td>
<td>Optional: Standard output filename</td>
</tr>
<tr>
<td><code>-e filename</code></td>
<td><code>--e filename</code></td>
<td>Optional: Error filename</td>
</tr>
<tr>
<td><code>-V</code></td>
<td><code>--export=all</code></td>
<td>(default)</td>
</tr>
<tr>
<td><code>-v var=&lt;value&gt;</code></td>
<td><code>--export=var</code></td>
<td>Optional: Export single environment variable var to job</td>
</tr>
<tr>
<td><code>-m be</code></td>
<td>`--mail-type=BEGIN</td>
<td>END</td>
</tr>
<tr>
<td><code>-M &lt;netid&gt;@email.arizona.edu</code></td>
<td><code>--mail-user=&lt;netid&gt;@email.arizona.edu</code></td>
<td>Optional: email address used for notifications</td>
</tr>
<tr>
<td><code>-l place=excl</code></td>
<td><code>--exclusive</code></td>
<td>Optional: Request exclusive access to node</td>
</tr>
</tbody>
</table>

### Environment Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$PBS_O_WORKDIR</code></td>
<td>Job submission directory</td>
</tr>
<tr>
<td><code>$SLURM_SUBMIT_DIR</code></td>
<td>Job submission directory</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 --account=&lt;PI GROUP&gt; #SBATCH --partition=standard #SBATCH --qos=user_qos_&lt;PI GROUP&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;PI GROUP&gt; 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>#SBATCH --array=1-2 #SBATCH -o %A.out #SBATCH --open-mode=append</td>
<td>12345.out</td>
</tr>
<tr>
<td>%a</td>
<td>A job array's index number</td>
<td>#SBATCH --array=1-2 #SBATCH -o %A_%a.out #SBATCH --open-mode=append</td>
<td>12345_1.out</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 #SBATCH -o %x.out</td>
<td>JobName.out</td>
</tr>
</tbody>
</table>
Job Examples

Single serial job submission

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 --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
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.
   ```
   #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

#!/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>
#PBS -J 1-5

cd $PBS_O_WORKDIR

pwd; hostname; date

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

SLURM Script

#!/bin/bash
#SBATCH --output=Sample_SLURM_Job-%a.out
#SBATCH --ntasks=1
#SBATCH --mem=1gb
#SBATCH --time=00:01:00
#SBATCH --partition=standard
#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

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

export SBATCH_GET_USER_ENV=1
export OMPI_MCA_btl_openib_opc_include=rdmacm
export OMPI_MCA_btl_openib_if_include=bnxt_re1
export OMPI_MCA_btl_openib_roce_enable=1
export OMPI_MCA_btl=vader,self,openib
export OMPI_MCA_oob_tcp_if_include=eth1

For Intel MPI, these variables are set for you:

export I_MPI_FABRICS=shm:ofi
export FI_PROVIDER=verbs
export FI_VERBS_IFACE=eth1

PBS Script

#!/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
The document contains a bash script and a description of its purpose. The script is used to submit a job to a cluster using SLURM. The script sets up the environment, runs an MPI program, and captures timing information. The description explains that the SLURM script inherits the environment and the job does not need to change directories. The script is designed to be executed by a job scheduler like SLURM on a cluster. The file `rosetta_min.pdf` is also referenced in the document, indicating that it might be related to the content of the script or the project being described.