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 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:

| Compute Resources then Example Resource Requests |

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:

```bash
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:

```bash
interactive -a account_name
```

Are you using X11 forwarding?

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

Full usage:

```bash
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:

```bash
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:

```bash
#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.

**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:

```bash
#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:

```bash
#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.

<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> (note the upper case 'I')</td>
<td><code>srun &lt;options&gt; --pty bash -i 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**

<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 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>#PBS -l mem=&lt;N&gt;gb</code></td>
<td><code>#SBATCH --mem=&lt;N&gt;gb</code></td>
<td>Select N gb of memory per node</td>
</tr>
<tr>
<td><code>#PBS -l pcmem=&lt;N&gt;gb</code></td>
<td><code>#SBATCH --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>#PBS J N-M</code></td>
<td><code>#SBATCH --array=N-M</code></td>
<td>Array job submissions where N and M are integers</td>
</tr>
<tr>
<td><code>#PBS -l np100s=1</code></td>
<td><code>#SBATCH --gres=gpu:1</code></td>
<td>Optional: Request a GPU</td>
</tr>
<tr>
<td>#PBS arguments</td>
<td>#SBATCH arguments</td>
<td>Details</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------</td>
<td>---------</td>
</tr>
<tr>
<td>-N JobName</td>
<td>--job-name=JobName</td>
<td>Optional: Set job name</td>
</tr>
<tr>
<td>-j oe</td>
<td></td>
<td>Optional: Combine stdout and error</td>
</tr>
<tr>
<td>-o filename</td>
<td>--output=filename</td>
<td>Optional: Standard output filename</td>
</tr>
<tr>
<td>-e filename</td>
<td>--error=filename</td>
<td>Optional: Error filename</td>
</tr>
<tr>
<td>-V</td>
<td>--export=all</td>
<td>Optional: Export all environment variables to job</td>
</tr>
<tr>
<td>-m be</td>
<td>--mail-type=BEGIN</td>
<td>Optional: Request email notifications</td>
</tr>
<tr>
<td>-M &lt;netid&gt;@email.arizona.edu</td>
<td>--mail-user=&lt;netid&gt;@email.arizona.edu</td>
<td>Optional: email address used for notifications</td>
</tr>
<tr>
<td>-l place=excl</td>
<td>--exclusive</td>
<td>Optional: Request exclusive access to node</td>
</tr>
</tbody>
</table>

Environment Variables

- $PBS_O_WORKDIR: Job submission directory
- $PBS_JOBID: Job ID
- $PBS_JOBNAME: Job name
- $PBS_ARRAY_INDEX: Index to differentiate tasks in an array
- $PBS_NODEFILE: List of nodes allocated to current job

Terminology

Queue

Group List

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>
<tbody>
<tr>
<td>standard</td>
<td>#SBATCH --account=&lt;PI GROUP&gt;</td>
<td>Consumes your group's standard allocation.</td>
</tr>
<tr>
<td></td>
<td>#SBATCH --partition=standard</td>
<td></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>
</tbody>
</table>
Available for groups who have purchased compute resources. The partition flag is left as standard and requires the additional qos flag. Replace <PI GROUP> with your group's name.

### 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</td>
<td>12345_1.out 12345_2.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**

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

cd $PBS_O_WORKDIR
pwd; hostname; date

module load python
python --version
```

**SLURM Script**
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 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

```bash
#!/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
python --version
```
MPI Example

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

```bash
export SBATCH_GET_USER_ENV=1
export OMP_MCA_btl_openib_cpc_include=rdmacm
export OMP_MCA_btl_openib_if_include=bnxt_re1
export OMP_MCA_btl_openib_roce_enable=1
export OMP_MCA_btl=vader,sel,openib
export OMP_MCA_oob_tcp_if_include=eth1
```

For **Intel MPI**, these variables are set for you:

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

PBS Script

```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 load openmpi
/usr/bin/time -o mpit_prog.timing mpirun -np 16 a.out
```

SLURM Script

```bash
#!/bin/bash
#SBATCH --output=Sample_SLURM_Job-%a.out
#SBATCH --ntasks=1
#SBATCH --nodes=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"
#!/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 Inheres your environment. cd $SLURM_SUBMIT_DIR not needed
cd
pwd; hostname; date
module load openmpi3
/usr/bin/time -o mpit_prog.timing mpirun -np 16 a.out

rosetta_min.pdf