Penn State Health HPC Slurm Tutorial

This is a basic tutorial explaining how to use the Slurm job management system that is integrated with the HPC environment. The purpose of this tutorial is to give a quickstart guide to those who are new to the Slurm environment on the HPC. If you have any additional questions regarding this tutorial you may use the following form.

Table of Contents

What is Slurm

Slurm is an open source, fault-tolerant, highly scaleable cluster management, and job scheduling system for large and small Linux clusters. Slurm has three key functions:

1. Allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work.
2. Provides a framework for starting, executing, and monitoring work (normally a parallel job) on the allocated nodes.
3. Arbitrates contention for resources by managing a queue of pending work. Optional plugins for: accounting, advanced reservation, gang scheduling (time sharing for parallel jobs), backfill scheduling, topology optimized resource selection, resource limits by user or account, and sophisticated multifactor job prioritization

System Setup

macOS Users

Those who have Macs should use the integrated Terminal application to connect to the HPC. If you are planning on using GUI applications on the HPC via X11 Forwarding, then you should also install XQuartz. When connecting to the HPC the additional flags -XY are needed.

For example: `ssh –XY user@hpcgateway7.pennstatehealth.net`

Windows Users

Windows users will have to download a SSH client in order to connect to the HPC. There are a multitude of applications that can do this easily, such as MobaXterm or PuTTY. Our team highly recommends that users use MobaXterm as it is more user-friendly for those that don't have prior experience using HPC.

Windows users should connect to the host `hpcgateway7.pennstatehealth.net` with their SSH clients.

Slurm at Penn State Health

System Configurations

Penn State Health Information Services maintains a Slurm cluster on the HPC. Currently, the Slurm cluster has 35 general purpose compute nodes, 3 GPU nodes, and 5 high-density GPU compute nodes. The specifications of the servers are as follows:

	Compute	GPU	High-Density

	Compute	GPU	High-Density
CPU	44 cores/88 threads	44 cores/88 threads	44 cores/88 threads
RAM	384 GB	384 GB	1.5 TB
GPU	None	2x Nvidia P100 16GB	4x Nvidia V100 32GB

These three types of nodes are broken down into three individual queues (called partitions) inside of Slurm. This info can be seen by using the `sinfo` command on the HPC.

```
PARTITION    AVAIL  TIMELIMIT  NODES  STATE NODELIST
compute         up   infinite     35   idle psh01com1hcom[01-35]
gpu             up   infinite      3   idle psh01com1hgpu[01-03]
dense           up   infinite      5   idle psh01com1hdns[01-05]
```

Default Parameters

Users submitting jobs without specifying resource allocations will have their jobs run with a default Slurm configuration. The following default values are set on the new Slurm cluster:

- Threads - 1 thread per job
- RAM - 8 GB RAM per job
- Job arrays - 10,000 tasks maximum per array
- Maximum job time - 3 days per job

All of these defaults can be overwritten with their respective parameters in salloc and sbatch. Currently, there are no restrictions on how many nodes or how much of a single node can be allocated.

Slurm Job Examples

The sample Slurm jobs used in this tutorial can be found on the HPC under the directory `/ri/shared/examples/SlurmExamples/`.

Basic Job

This job is the most basic job that you can run in Slurm. Examples later in the tutorial will build off of the concepts used in this job.

The following file can be found at `/ri/shared/examples/SlurmExamples/SlurmBasicJob.sbatch`:

```
#!/bin/bash -l
#SBATCH --job-name=BasicJob
#SBATCH --output=BasicJob_results.%j.%N.txt
#SBATCH --error=BasicJob_errors.%j.%N.err
#SBATCH -p compute
#SBATCH -A <myaccount>

srun hostname
```

This job file uses a few directives to tell Slurm how to handle the job. It tells it to:

1. Set the job name to "BasicJob"
2. Write terminal output to the file "BasicJob_results.[job number].[node name].txt"
3. Write error output to the file "BasicJob_errors.[job number].[node name].err"
4. Run the job on the "compute" partition
5. Charge resources used in the job to the account "myaccount"

After the directives, commands can be launched on the node assigned to the job via the "srun" command. If you run the command `hostname`, you will see the hostname of the server you are currently logged into. The command `srun hostname` will print out the hostname of the server that processed the job.

To submit this job, run the command `sbatch SlurmBasicJob.sbatch` in your terminal.

After the job runs you will see that the results file contains the hostname of the server that ran your job.

Cancelling A Job

Sometimes it is necessary to kill a Slurm job. In this example a job is submitted to the cluster to run at a later time but needs to be cancelled by the user before it runs.

The following file can be found at `/ri/shared/examples/SlurmExamples/SlurmFutureJob.sbatch`:

```
#!/bin/bash -l
#SBATCH --job-name=FutureJob
#SBATCH --output=FutureJob_results.%j.%N.txt
#SBATCH --error=FutureJob_errors.%j.%N.err
```

```
#SBATCH --begin=now+1hour
#SBATCH -p compute
#SBATCH -A <myaccount>

srun hostname
```

This job will execute an hour after it is submitted to the cluster with the command `sbatch SlurmFutureJob.sbatch`. This is denoted by the `--begin` directive in the job submission file. When the job is submitted a job ID number is created by the cluster. The typical way to kill a job is to use the command `scancel <job_id>`.

For example, if a job with ID 123456 is created when the job is submitted then the corresponding scancel command would be `scancel 123456`.

Jobs can also be cancelled by specifying the user that created them. For example, the command `scancel -u cke5045` would cancel all jobs created by the user cke5045. However, users will not be able to cancel jobs submitted by other users of the cluster.

Basic Job With Email Notifications

Slurm now has the ability to email you of changes in your job status. The following example builds off of our earlier basic job.

The following file can be found at `/ri/shared/examples/SlurmExamples/SlurmBasicEmailJob.sbatch`:

```
#!/bin/bash -l
#SBATCH --job-name=BasicEmailJob
#SBATCH --output=BasicEmailJob_results.%j.%N.txt
#SBATCH --error=BasicEmailJob_errors.%j.%N.err
#SBATCH -p compute
#SBATCH -A <myaccount>
#SBATCH --mail-user=<user1>@pennstatehealth.psu.edu,
<user2>@pennstatehealth.psu.edu
#SBATCH --mail-type=ALL

srun hostname
```

This job makes use of the --mail-user and --mail-type directives. What this will do is send users email notifications of changes in the job status. For example, after submitting the job with the command `sbatch SlurmBasicEmailJob.sbatch` the specified users will receive a notification that the job has been submitted to the cluster. The users will also receive an email when the job finishes with a small summary of statistics about that job.

--mail-user

The flag --mail-user is a comma separated list of users to send emails to.

--mail-type

The flag --mail-type defines what types of emails will be sent to the user. Here, the value "ALL" is used so that the user is notified of all changes in the jobs submitted. From the official Slurm documentation:

```
--mail-type=<type>
    Notify user by email when certain event types occur. Valid type values are
NONE, BEGIN, END, FAIL, REQUEUE, ALL (equivalent to BEGIN, END, FAIL, REQUEUE,
and STAGE_OUT), STAGE_OUT (burst buffer stage out and teardown completed),
TIME_LIMIT, TIME_LIMIT_90 (reached 90 percent of time limit), TIME_LIMIT_80
(reached 80 percent of time limit), TIME_LIMIT_50 (reached 50 percent of time
limit) and ARRAY_TASKS (send emails for each array task). Multiple type values
may be specified in a comma separated list. The user to be notified is
indicated with --mail-user. Unless the ARRAY_TASKS option is specified, mail
notifications on job BEGIN, END and FAIL apply to a job array as a whole rather
than generating individual email messages for each task in the job array.
```

Basic Job With System Resources Defined

Should a job need more than the default allocation, users can supply their own job allocation requirements. There are a number of flags that can be set in sbatch/salloc.

The following file can be found at `/ri/shared/examples/SlurmExamples/SlurmSystemResourcesJob.sbatch`:

```
#!/bin/bash -l
#SBATCH --job-name=BasicSystemResourcesJob
#SBATCH --output=BasicSystemResourcesJob_results.%j.%N.txt
#SBATCH --error=BasicSystemResources_errors.%j.%N.err
#SBATCH -p compute
#SBATCH -A <myaccount>
#SBATCH --nodes 2
#SBATCH --ntasks 2
#SBATCH --cpus-per-task 16
#SBATCH --mem-per-cpu=1000

srun hostname
```

Submit this job to the cluster with the command `sbatch SlurmSystemResourcesJob.sbatch`.

This example includes the addition of four new flags. There are many flags that can be used to define allocation requirements, but these are a few of the most

commonly used flags. Here is a brief explanation of each flag used in this example from the official Slurm documentation.

--nodes

```
-N, --nodes=<minnodes[-maxnodes]>
    Request that a minimum of minnodes nodes be allocated to this job. A
maximum node count may also be specified with maxnodes. If only one number is
specified, this is used as both the minimum and maximum node count. The
partition's node limits supersede those of the job. If a job's node limits are
outside of the range permitted for its associated partition, the job will be
left in a PENDING state. This permits possible execution at a later time, when
the partition limit is changed. If a job node limit exceeds the number of nodes
configured in the partition, the job will be rejected. Note that the
environment variable SLURM_JOB_NODES will be set to the count of nodes actually
allocated to the job. See the ENVIRONMENT VARIABLES section for more
information. If -N is not specified, the default behavior is to allocate enough
nodes to satisfy the requirements of the -n and -c options. The job will be
allocated as many nodes as possible within the range specified and without
delaying the initiation of the job. The node count specification may include a
numeric value followed by a suffix of "k" (multiplies numeric value by 1,024)
or "m" (multiplies numeric value by 1,048,576).
```

--ntasks

```
-n, --ntasks=<number>
    sbatch does not launch tasks, it requests an allocation of resources and
submits a batch script. This option advises the Slurm controller that job steps
run within the allocation will launch a maximum of number tasks and to provide
for sufficient resources. The default is one task per node, but note that the -
-cpus-per-task option will change this default.
```

--cpus-per-task

```
-c, --cpus-per-task=<ncpus>
    Advise the Slurm controller that ensuing job steps will require ncpus
number of processors per task. Without this option, the controller will just
try to allocate one processor per task.

    For instance, consider an application that has 4 tasks, each requiring 3
processors. If our cluster is comprised of quad-processors nodes and we simply
ask for 12 processors, the controller might give us only 3 nodes. However, by
using the --cpus-per-task=3 options, the controller knows that each task
requires 3 processors on the same node, and the controller will grant an
allocation of 4 nodes, one for each of the 4 tasks.
```

--mem-per-cpu

```
--mem-per-cpu=<size[units]>
    Minimum memory required per allocated CPU. Default units are megabytes
unless the SchedulerParameters configuration parameter includes the
"default_gbytes" option for gigabytes. Default value is DefMemPerCPU and the
maximum value is MaxMemPerCPU (see exception below). If configured, both
parameters can be seen using the scontrol show config command. Note that if the
```

```
job's --mem-per-cpu value exceeds the configured MaxMemPerCPU, then the user's
limit will be treated as a memory limit per task; --mem-per-cpu will be reduced
to a value no larger than MaxMemPerCPU; --cpus-per-task will be set and the
value of --cpus-per-task multiplied by the new --mem-per-cpu value will equal
the original --mem-per-cpu value specified by the user.
```

Basic Job With GPUs

Users can now request allocations of GPUs inside Slurm. The following is a basic example to request a GPU.

The following file can be found at `/ri/shared/examples/SlurmExamples/SlurmGPUJob.sbatch`:

```
#!/bin/bash -l
#SBATCH --job-name=GPUJob
#SBATCH --output=GPUJob_results.%j.%N.txt
#SBATCH --error=GPUJob_errors.%j.%N.err
#SBATCH -p gpu,dense
#SBATCH -A <myaccount>
#SBATCH --gres=gpu:1

srun nvidia-smi
```

After running the job with the command `sbatch SlurmGPUJob.sbatch`, the output of `nvidia-smi` is printed out to the log file. This is a basic test in which you can see that the available GPU and its load.

The above example only shows one possible GRES value to allocate a GPU. On the HPC systems there are a few other possible values that can be used. The following examples use the format of gpu:[type]:[number] to allocate GPUs to a job, where [type] is the type of GPU and [number] is the number of GPUs allocated:

- gpu:1
- gpu:p100:2
- gpu:v100:4

That is a small set of possible values. Actual type and number of GPUs depends on the partition and system configuration as discussed in the System Configurations section.

Job Arrays

One popular way of launching jobs inside of Slurm is through job arrays. Job arrays allow for the submission of a large number of independent jobs under a

single job. They are commonly used when working with either similar or the same datasets.

The following file can be found at `/ri/shared/examples/SlurmExamples/SlurmJobArray.sbatch`:

```
#!/bin/bash -l
#SBATCH --job-name=SlurmJobArray
#SBATCH --output=SlurmJobArray_results.%j.%N.txt
#SBATCH --error=SlurmJobArray_errors.%j.%N.err
#SBATCH -p compute
#SBATCH -A <myaccount>
#SBATCH --array=1-10

echo "This is Slurm array task #" $SLURM_ARRAY_TASK_ID
```

Submit the job with the command `sbatch SlurmJobArray.sbatch`. In the results file there should be a number of lines from each task stating what task number corresponded to that message. While this output itself is not particulary useful, it demonstrates the concept that could be used in more complex jobs with large amounts of data.

Basic Interactive Job

Interactive jobs are a good alternative for users that are not comfortable or familiar with writing job files. Users can request an allocation of node(s) with the `salloc` command. For example, you can request a job on the compute partition with the command `salloc -p compute -A <myaccount>`. After the allocation is successful, you can run commands on the remote node with the `srun` command. For example, the command `srun hostname` will print the hostname of the node(s) processing the job. To exit/relinquish the job allocation simply enter the `exit` command into your session, or use the `Control + d` key combination.

Note: Interactive jobs will only work properly if they are launched within a SSH session with X11 Forwarding enabled. For more information on how to do this, see the System Setup section.

Interactive Job With X11 Forwarding

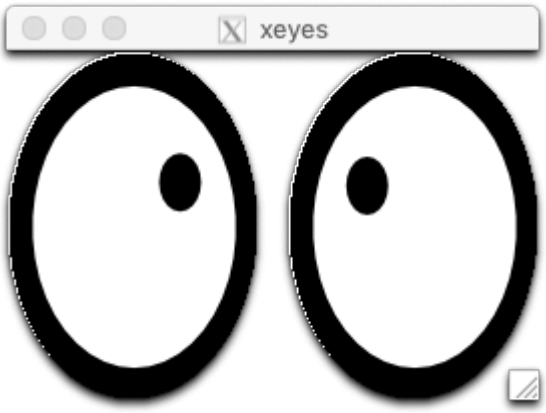
Users that need to launch GUI applications on the HPC will need to do so through interactive jobs with X11 Forwarding enabled. The command used is the same as a normal interactive job with the addition of the --x11 flag. To test this functionality, run the following commands:

```

```

```
salloc -p compute -A <myaccount> --x11
[Wait until allocation is granted]
srun xeyes
```

After running the second command, the xeyes GUI should load. This is just a simple test application to ensure that X11 Forwarding is working properly in your session. Once the application is closed, exit the job allocation again with either the `exit` command or the `Control + d` key combination.



Launching an Interactive Remote Program

On the HPC there are several interactive applications (R, Miniconda, Perl, Python) that may require additional user input while they are running. However, when launching these applications through Slurm's interactive mode you may run into unexpected errors from `srun`. These errors occur because the default behavior Slurm expects to see is a program run and exit, but these applications instead run and then wait for user input.

The fix to this is to include the `--pty` flag in your `srun` commands. The `--pty` flag tells srun to launch the command on the remote compute node in a pseudo terminal. This allows the compute node to wait for further input from the user and will only exit when the user closes the program.

There are two methods to prevent these errors when running interactive applications.

Launching a Remote Bash Session in a Pseudo Terminal

This method will launch a remote bash shell session on the compute node. This allows users to control a server without needing to use `srun` with every command launched on the compute server.

First, you must request an allocation through Slurm. This can be done by entering the following command to get a basic allocation:

```
salloc -p compute
```

You will see salloc output similar to this:

```
salloc: Granted job allocation <JOB ID>
salloc: Waiting for resource configuration
salloc: Nodes <HOSTNAME> are ready for job
```

Once your node is ready you can launch a pseudo terminal on the compute node with the command `srun --pty bash`.

After launching your pseudo terminal on the remote node you will see your prompt switch from the gateway node to that of the hostname from the previous salloc command output. For example:

- [cke5045@psh01com1hgtw02 ~]$ will switch to [cke5045@psh01com1hcom30 ~]$
- [rac42@psh01com1hgtw01 ~]$ will switch to [rac42@psh01com1hcom14 ~]$

All subsequent commands will be launched on the remote compute node once the prompt switches.

When finished, exit your Slurm allocation by entering the `exit` command or using the key combination `ctrl+d` until you see the following message:

```
salloc: Relinquishing job allocation <JOB ID>
```

For example, the following commands can be used to launch a new R shell inside a remote bash session:

```
> salloc -p compute

salloc: Granted job allocation <JOB ID>
salloc: Waiting for resource configuration
salloc: Nodes psh01com1hcom01 are ready for job

> srun --pty bash
> module load R
> R
> exit
> exit
```

After you are finished, please ensure that you relinquish your Slurm allocation with the `exit` commands so that you are no longer being charged for compute time.

Launching a Remote Program in a Pseudo Terminal

This method is similar to the previous one, but does not make use of the remote bash session. This method allows you to launch commands on the remote server without launching a remote bash shell session.

As with previous examples, request an allocation through Slurm:

`salloc -p compute`

You will see output similar to this once the allocation is granted:

```
salloc: Granted job allocation <JOB ID>
salloc: Waiting for resource configuration
salloc: Nodes <HOSTNAME> are ready for job
```

Once your node is ready you can execute commands on the remote server using `srun`.

For example, the following commands can be used to launch a new R shell on a remote compute server:

```
> salloc -p compute

salloc: Granted job allocation <JOB ID>
salloc: Waiting for resource configuration
salloc: Nodes psh01com1hcom01 are ready for job

> module load R
> srun --pty R
> exit
```

After you are finished, please ensure that you relinquish your Slurm allocation with the `exit` commands so that you are no longer being charged for compute time.

Useful Slurm Commands and Parameters

Additional resources and parameters can be found at the following links:

- Slurm sinfo documentation
- Slurm squeue documentation
- Slurm sbatch documentation
- Slurm salloc documentation
- Slurm sstat documentation
- Slurm scontrol documentation

sinfo

sinfo is used to view partition and node information for a system running Slurm.

```
sinfo [OPTIONS...] examples:
[-a, --all] Display information about all partitions.
```

squeue

squeue is used to view job and job step information for jobs managed by Slurm.

```
squeue [OPTIONS...] examples:

[-A <account_list>, --account=<account_list>] Specify the accounts
of the jobs to view. Accepts a comma separated list of account
names.
[-a, --all] Display information about jobs and job steps in all
partitions.
```

sbatch

sbatch is used to submit a job script to Slurm through:

- a file name on the command line
- standard input if no file name is given

```
sbatch [options] script [args...]   examples:

-A, --account=<account> Charge resources used by this job to
specified account. The account is an arbitrary string.
-n, --ntasks=<number> sbatch does not launch tasks, it requests an
allocation of resources and submits a batch script. This option
advises the Slurm controller that job steps run within the
allocation will launch a maximum of number tasks and to provide for
sufficient resources.
-c, --cpus-per-task=<ncpus> Advise the Slurm controller that
ensuing job steps will require ncpus number of processors per task.
-d, --dependency=<dependency_list> Defer the start of this job
until the specified dependencies have been satisfied completed.
-D, --workdir=<directory>
--mem=<MB> Specify the real memory required per node
--mem-per-cpu=<MB> Mimimum memory required per allocated CPU in
MegaBytes.n MegaBytes
--mail-user=<user>
--mail-type=<type> (example ,END,FAIL,ALL among others. Multiple
can be added comma delimited).
-a, --array=<indexes>Submit a job array, multiple jobs to be
executed with identical parameters. The indexes specification
identifies what array index values should be used. Multiple values
```

```
may be specified using a comma separated list and/or a range of
values with a "-" separator. For example, "--array=0-15" or "--
array=0,6,16-32".
```

salloc

salloc is used to create node allocations for interactive mode.

```
salloc [options] script [args...] examples:

-A, --account=<account> Charge resources used by this job to
specified account. The account is an arbitrary string.
-n, --ntasks=<number> sbatch does not launch tasks, it requests an
allocation of resources and submits a batch script. This option
advises the Slurm controller that job steps run within the
allocation will launch a maximum of number tasks and to provide for
sufficient resources.
-c, --cpus-per-task=<ncpus> Advise the Slurm controller that
ensuing job steps will require ncpus number of processors per task.
-d, --dependency=<dependency_list> Defer the start of this job
until the specified dependencies have been satisfied completed.
-D, --workdir=<directory>
--mem=<MB> Specify the real memory required per node
--mem-per-cpu=<MB> Mimimum memory required per allocated CPU in
MegaBytes.n MegaBytes
--mail-user=<user>
--mail-type=<type> (example ,END,FAIL,ALL among others. Multiple
can be added comma delimited).
```

sstat

sstat is used to display a variety of status information about a running job or step.

Examples:

```
sstat --format=AveCPU,AvePages,AveRSS,AveVMSize,JobID -j 11
sstat -p --format=AveCPU,AvePages,AveRSS,AveVMSize,JobID -j 11
```

Requesting Help

The HPC team now has simplified forms that can be used for HPC requests. These forms help to ensure that your request gets routed to the correct people and is addressed promptly.

Software Request

If there is software that is not currently installed on the HPC but you would like it to be, you can use the following form to request that it be installed.

Help Request

If you have any other questions,issues, or concerns that you would like the HPC team to address, you can use this form to contact them. Here is a list of information (if applicable) that will help the team process the request faster:

- What node are you on?
- What was the job ID of the job?
- Where is your job submission file located?

Additional Resources

If you are still looking to learn more about Slurm, you can find additional documentation and tutorials at the following links:

- Slurm Suggested Tutorials
- NERSC Sample Jobs
- UNamur Slurm Tutorial
- LLNL Slurm Quick Start Guide
- Bright Computing Slurm 101 Guide