Ginsburg - Submitting Jobs

- Processes and Jobs
- Restrictions on Login Node Usage
- Core-Based Jobs
- Node-Based Jobs
- Interactive Jobs
- Basic Job Directives
- Walltime
- Memory Requests
- Job Arrays
- Job scheduling basics
- Job limits
- Slurm Quick Start Guide
- Lustre Background and Basics

Processes and Jobs

In computing, a process is an instance of a computer program that is being executed. It contains the program code and its current activity. A process is normally launched by invoking it by the name of the executable (compiled code) associated with it, either directly at the Unix shell prompt, or within a shell script.

A job in Slurm (the scheduler on Ginsburg) is an allocation of resources assigned to a user for a specified amount of time. Within a job, process or processes can be launched: for batch jobs that is done in the submit script used for running the job, and for interactive jobs at the shell prompt that appears on running one. All such processes execute on the compute (aka “execute”) nodes, which are dedicated precisely to that purpose.

Restrictions on Login Node Usage

After logging in, you land on a login (aka “head”) node, from which users normally launch their jobs. The login node has some restrictions on the scope of processes that can be run on it. In order to allow for special projects and activities, these restrictions are still quite lenient at this time, and we strongly rely on our users to severely limit launching processes on the head node.

When a user ignores that recommendation and executes processes that are compute-intensive, longer than momentary, and especially requiring multiple cores, the login node becomes overloaded, preventing other users from doing their regular work. In such cases, we typically terminate the processes and inform the user about it with a request to run the processes within jobs instead. Please be aware that the cluster is a shared resource and cooperate with us on trying to limit the computing activity on the head node to minimum.

If you need to run a CPU intensive process, please start an interactive job as described bellow:

https://confluence.columbia.edu/confluence/display/rcs/Ginsburg+-+Submitting+Jobs#Ginsburg-SubmittingJobs-InteractiveJobs

Jobs can request compute resources on a per-core basis or a per-node basis.

Core-Based Jobs

Jobs wishing to use less than a full node should specify the number of cores required. If you're not sure how many cores to request then 1 is most likely to be the correct number to use. The maximum number of cores available on a node is 32.

Cores can be requested using either -c or --cpus-per-task (in this and the following examples, “or” indicates an exactly equivalent alternative syntax).

```
#SBATCH --cpus-per-task=1
```

or

```
#SBATCH --cpus-per-task=1
```
It is important to also specify your memory requirement when using less than a full node as this will allow the scheduler to ensure that there will be enough memory available on the node where your job runs.

Node-Based Jobs

Jobs can also request more than one node. This is often the most efficient way to run parallel distributed jobs, such as those using MPI.

To specify the number of nodes, use -N or --nodes.

```
#SBATCH -N 1
```

or

```
#SBATCH --nodes=1
```

You should may specify a memory requirement when requesting use of a node.

```
#SBATCH --mem=187G    # Standard nodes have approximately 187G of total memory available
```

To explicitly request a "standard node" with 192 GB RAM, you may specify the mem192 feature.

```
#SBATCH -C mem192
```

If your job requires more than the standard 192 GB then you may optionally add a constraint to request one of the cluster's high memory nodes, each of which has 768 GB of memory. The feature to request is "mem768".

```
#SBATCH -C mem768
```

or

```
#SBATCH --constraint=mem768
```

Please keep in mind that the above directives will only secure the corresponding type of a node but do not ensure that all its memory is available to the job (even if one specifies its -exclusive usage). By default, 5,800 MB is allocated to each core.

Interactive Jobs

Interactive jobs allow user interaction during their execution. They deliver to the user a new shell from which applications can be launched.
To submit an interactive job, run the following, where "<ACCOUNT>" is your group’s account name.

```
srun --pty -t 0-01:00 -A <ACCOUNT> /bin/bash
```

The Slurm directives noted below that begin with #SBATCH are available on the command line as well for interactive jobs.

For example, to run an interactive job on a large node, use:

```
srun --pty -t 0-01:00 -C mem768 -A <ACCOUNT> /bin/bash
```

If a node is available, it will be picked for you automatically, and you will see a command line prompt on a shell running on it. If no nodes are available, your current shell will wait.

### Basic Job Directives

The following table lists some common directives used in Slurm submit scripts. Each should be preceded by #SBATCH when used in a submit script. Many directives have a short alternate name and these are listed where available. The examples sometimes use the long version of a given directive and sometimes the short version; in either case no preference is implied.

<table>
<thead>
<tr>
<th>Directive</th>
<th>Short Version</th>
<th>Description</th>
<th>Example</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>--account=&lt;account&gt;</td>
<td>-A &lt;account&gt;</td>
<td>Account.</td>
<td>#SBATCH --account=stats</td>
<td></td>
</tr>
<tr>
<td>--job-name=&lt;job name&gt;</td>
<td>-J &lt;job name&gt;</td>
<td>Job name.</td>
<td>#SBATCH -J DiscoProject</td>
<td></td>
</tr>
<tr>
<td>--time=&lt;time&gt;</td>
<td>-t &lt;time&gt;</td>
<td>Time required.</td>
<td>#SBATCH --time=10:00:00</td>
<td>The maximum time allowed is five days.</td>
</tr>
<tr>
<td>--mem=&lt;memory&gt;</td>
<td>-m &lt;memory&gt;</td>
<td>Memory required per node.</td>
<td>#SBATCH --mem=16gb</td>
<td></td>
</tr>
<tr>
<td>--mem-per-cpu=&lt;memory&gt;</td>
<td>-M &lt;memory&gt;</td>
<td>Memory per cpu.</td>
<td>#SBATCH --mem-per-cpu=5G</td>
<td></td>
</tr>
<tr>
<td>--constraint=mem768</td>
<td>-C mem768</td>
<td>Specifying a large (768 GB RAM) node.</td>
<td>#SBATCH -C mem768</td>
<td></td>
</tr>
<tr>
<td>--cpus-per-task=&lt;cpus&gt;</td>
<td>-c &lt;cpus&gt;</td>
<td>CPU cores per task.</td>
<td>#SBATCH -c 1</td>
<td>Nodes have a maximum of 24 cores.</td>
</tr>
<tr>
<td>--nodes=&lt;nodes&gt;</td>
<td>-N &lt;nodes&gt;</td>
<td>Nodes required for the job.</td>
<td>#SBATCH -N 4</td>
<td></td>
</tr>
<tr>
<td>--array=&lt;indexes&gt;</td>
<td>-A &lt;indexes&gt;</td>
<td>Submit a job array.</td>
<td>#SBATCH -a 1-4</td>
<td>See below for discussion of job arrays.</td>
</tr>
<tr>
<td>--mail-type=&lt;ALL,BEGIN,END,FAIL,NONE&gt;</td>
<td>-m &lt;mail type&gt;</td>
<td>Send email job notifications</td>
<td>#SBATCH --mail-type=ALL</td>
<td></td>
</tr>
<tr>
<td>--mail-user=&lt;email_address&gt;</td>
<td>-u &lt;email&gt;</td>
<td>Email address</td>
<td>#SBATCH --mail-user=<a href="mailto:me@email.com">me@email.com</a></td>
<td></td>
</tr>
</tbody>
</table>

### Walltime

The walltime is specified with "-t" flag. For example:
#SBATCH -t 10:00:00
That is walltime format that translates to 10 hours (00 minutes and 00 seconds).
If you want to request just 1 hour walltime, you should request 1:00:00

Acceptable time formats in Slurm scheduler are: "minutes", "minutes:seconds", "hours:minutes:seconds",
"days-hours", "days-hours:minutes" and "days-hours:minutes:seconds".
The maximum time allowed is five days.

Memory Requests

There are two ways to ask for memory and they are mutually exclusive. You can ask either for
1) memory per cpu
or
2) memory per node

If you do not specify the memory requirement, by default you get 5,800 MB per CPU.

Ginsburg has 87 Standard Nodes with 192 GB of memory (about 187 GB usable).

Ginsburg’s 30 high memory nodes have 768 GB of memory each. They are otherwise identical to the standard nodes.

For example,
--mem-per-cpu=5gb
Minimum memory required per allocated CPU. If you request 32 cores (one node) you will get 160gb of memory on both standard node and on high-memory node.

If you specify the real memory required per node:
--mem=160gb
You will get the same.

However, if you specify
#SBATCH --exclusive
#SBATCH -C mem768
#SBATCH --mem=700gb
You will get 700 gb on high memory node.

Job Arrays

Multiple copies of a job can be submitted by using a job array. The --array option can be used to specify the job indexes Slurm should apply.

An existing submit file can be used to submit a job array by adding the flag to the sbatch command line.

$ sbatch -a 1-5 helloworld.sh
Submitted batch job 629249

In this example the job IDs will be the number 629249 followed by _1, _2, etc. so the first job in the array can be accessed using the job ID 629249_1 and the last 629249_5.

$ scontrol show job 629249_1

Note: There is a limit of 1,001 max job elements in an job array. If you try to submit more than 1,001 elements, the scheduler issues the following:
"Batch job submission failed: Invalid job array specification".
The environment variable $SLURM_ARRAY_TASK_ID indicates the index of the array element (i.e. job) in the job array, and is accessible from within that job.

**Job scheduling basics**

The walltime limit on the cluster is **5 days (120 hours)**. 5 day jobs may only be run on nodes that your group owns.

For all other nodes, the time limit is **12 hours**.

You have priority access to the nodes owned by your group. That implies that the maximum time a job from your group will need to wait for one of those nodes is 12 hours (and that occurs only in one of those rare case when a job by someone from another group starts running on such a node just before you or someone else in your group launch yours).

In order to guarantee that, users are allowed to run on other groups’ nodes only for maximum of 12 hours.

**Job limits**

*Note: There is a limit of 1,005 max jobs running per user. Any jobs that exceed this limit will remain in queue until (some of) the user’s other running jobs complete.*

Additionally, there is a **limit of 5,000 max submitted jobs** per user simultaneously. If you try to submit more than 5,000 jobs simultaneously, the scheduler displays the following error:

```
sbatch: error: Batch job submission failed: Job violates accounting/QOS policy (job submit limit, user’s size and/or time limits)
```

**Slurm Quick Start Guide**

See the Slurm Quick Start Guide for a more in-depth introduction on using the Slurm scheduler.

**Lustre Background and Basics**

The Ginsburg cluster utilizes Lustre, which is a robust file system that consists of servers and storage. A Metadata Server (MDS) tracks metadata (for example, ownership and permissions of a file or directory). Object Storage Servers (OSSs) provide file I/O services for Object Storage Targets (OSTs), which host the actual data storage. An OST is typically a single disk array. A notional diagram of a Lustre file system is shown in Figure 1, with one MDS, three OSSs, and two OSTs per OSS for a total of six OSTs. A Lustre parallel file system achieves its performance by automatically partitioning data into chunks, known as “stripes,” and writing the stripes in round-robin fashion across multiple OSTs. This process, called “striping,” can significantly improve file I/O speed by eliminating single-disk bottlenecks.

For jobs generating a large amount of I/O requests which are relatively small in size, e.g., jobs using `bedtools`, `Ancestry HMM` or even Matlab, setting a default stripe count to "1" is recommended.

```
$ mkdir workdir001
$ lfs setstripe -c 1 workdir001
```

In the above example we are first creating a directory called "workdir001" and then setting a striping policy of 1. Any files you create under `workdir001` will inherit the new striping policy. Subdirectories will not inherit the striping policy. The subdirectories will inherit the striping policy set on the parent directory if they are created after the striping policy is set.

**The lfs getstripe Command**
The "lfs getstripe" command reports the stripe characteristics of a file or directory.

Syntax:

Example:
$ lfs getstripe MyDir
MyDir
 stripe_count: 1 stripe_size: 1048576 stripe_offset: -1

The output shows that files created in the directory MyDir will be stored using one stripe of 1048576 bytes (1 MB) per block unless explicitly striped otherwise before writing. The stripe_offset (also known as stripe index) of -1 means that each file will have an OST placement determined automatically by Lustre.