Using Slurm Basic

This page gives a basic introduction for people who are NOT used to using LSF. (If you're used to using LSF, please consult the O2 - Moving from LSF to Slurm page instead.)

Introduction to the Introduction

Learning about a new computer system can be intimidating for some. Some biologists are unfamiliar with UNIX, the command line, and clusters. But don't worry! Literally thousands of biologists have learned how to use our resources. You only need to learn a few things to get started, and if you run into trouble, you can always contact Research Computing for help.

The O2 cluster is a collection of hundreds of computers with thousands of processing cores. SLURM (Simple Linux Utility for Resource Management) is basically a system for ensuring that the hundreds of users "fairly" share the processors and memory in the cluster.

The basic process of running jobs:

1. You login via SSH (secure shell) to the host: **`o2.hms.harvard.edu`**
   a. If you are connecting to O2 from **outside of the HMS network,** you will be required to use two-factor authentication.
   b. Please reference this wiki page for more information on how to login to the cluster.
2. If necessary, you copy your data to the cluster from your desktop or another location (see File Transfer). You may also want to copy large data inputs to the scratch filesystem (see Filesystems) for faster processing.
3. You submit your job - for example, a program to map DNA reads to a genome - specifying how long your job will take to run and what partition to run in. You can modify your job submission in many ways, like requesting a large amount of memory, as described below.
4. Your job sits in a partition (job status `PENDING`), and when it's your turn to run, SLURM finds a computer that's not too busy.
5. Your job runs on that computer (a "compute node"), and goes to status `RUNNING`. While it's running, you don't interact with the program. (If you're running a program that requires user input or pointing and clicking, see Interactive Sessions below.)
6. The job finishes running (status `COMPLETED`, or `FAILED` if it had an error). You get an email when the job is finished if you specified `--mail-type` (and optionally `--mail-user`) in your job submission command.
7. If necessary, you might want to copy data back from the scratch filesystem to a backed-up location, or to your desktop.

Definitions

Here's some of the terminology that you'll find used in relation to SLURM in this and other documents:

- **host**, **node**, and **server** are all just fancy words for a computer
- **core** Cluster nodes can run multiple jobs at a time. Currently, the nodes on O2 have 32 cores, meaning they can run 32 simple jobs at a time. (Some jobs use more than one core.)
- **master host:** The system that performs overall coordination of the SLURM cluster. In our case we have a primary server named `slurm-`

`prod01` and one backup.

- **submission host:** When logging in to `o2.hms.harvard.edu`, you'll actually end up on a "login node" called `login01`, `login02`, `login03`, `login04` or `login05`. You submit your SLURM jobs from there.
- **execution host:** (or "compute node") A system where SLURM jobs will actually run. In O2, compute nodes are named `compute-x-yy-zz`. x, yy and zz tell the location of the machine in the computer room.
- **partition:** The basic container for SLURM jobs. Partition tell the type of jobs that can be run through them, what resources those jobs can access, who can submit jobs to a given partition, and so forth.
- **filesystem:** From your perspective, just a fancy word for a big disk drive. The different filesystems O2 sees have different characteristics, in terms of the speed of reading/writing data, whether they're backed up, etc. See Filesystems.

Partitions (aka Queues in LSF)

The current partition configuration:

Partition	Specification *
short	12 hours
medium	5 days
long	30 days
interactive	2 job limit, 12 hours
mpi	5 days limit
priority	2 job limit, 30 day limit (same as LSF priority queue)
transfer	4 cores max, 5 days limit, 5 concurrently cores per user
gpu	see Using O2 GPU resources

check our How to choose a partition in O2 chart to see which partition you should use

***** For all partitions except for mpi, there is a maximum of 20 cores per job.

Basic SLURM commands

Note: Any time `<userid>` is mentioned in this document, it should be replaced with your eCommons ID (and omit the <>). Likewise, `<jobid>` should be replaced with an actual job ID, such as 12345. The name of a batch job submission script should be inserted wherever `<jobscript>` is mentioned.

SLURM command	Sample command syntax	Meaning
sbatch	`sbatch <jobscript>`	Submit a batch job.
srun	`srun --pty -t 0-0:5:0 -p interactive /bin/bash`	Start an interactive session for five minutes in the interactive queue.
squeue	`squeue -u <userid>`	View status of your jobs in the queue. Only non-completed jobs will be shown.
scontrol	`scontrol show job <jobid>`	Look at a running job in detail. For more information about the job, add the `-dd` parameter.
scancel	`scancel <jobid>`	Cancel a job. `scancel` can also be used to kill job arrays or job steps.
scontrol	`scontrol hold <jobid>`	Pause a job
scontrol	`scontrol release <jobid>`	Release a held job (allow it to run)
sacct	`sacct -j <jobid>`	Check job accounting data. Running `sacct` is most useful for completed jobs.

sinfo	`sinfo`	See node and partition information. Use the `-N` parameter to see information per node.

Submitting Jobs

There are two commands to submit jobs on O2: `sbatch` or `srun`.

`sbatch` is used for submitting batch jobs, which means you should write a script to use with this command. When invoked, `sbatch` creates a job allocation (resources such as nodes and processors) before running the commands specified in the script.

`srun` is used for starting job steps. It can be used from within a pre-existing allocation, such as within an `sbatch` script, or can create a new allocation.

Note: If you would like to submit a job with a script, use `sbatch`. If you want to start an interactive session, use `srun`.

The `sbatch` command

You submit jobs to SLURM using the `sbatch` command, followed by the script you'd like to run. When you submit the job, `sbatch` will give you a numeric JobID that you can later use to monitor your job. The SLURM scheduler will then find a computer that has an open slot matching any specifications you gave (see below on requesting resources), and tell that computer to run your job.

`sbatch` also accepts option arguments to configure your job, and O2 users should specify all of the following with each job, at a minimum:

- the partition (using `-p`)
- a runtime limit, i.e., the maximum hours and minutes (`-t 2:30:00`) the job will run. The **job will be killed if it runs longer than this limit**, so it's better to overestimate.

Most users will be submitting jobs to the `short,medium, long`, `priority`, or `interactive` partitions, some in `mpi`. Here is a guide to choose a proper partition:

Workflow	Best queue to use	Time limit
1 or 2 jobs at a time	`priority`	1 month
> 2 short jobs	`short`	12 hours
> 2 medium jobs	`medium`	5 days
> 2 long jobs	`long`	1 month
MPI parallel job using multiple nodes (with sbatch `-a` and `mpirun.sh`)	`mpi`	5 days
Interactive work (e.g. MATLAB graphical interface, editing/testing code) rather than a batch job	`interactive`	12 hours

`sbatch` options quick reference

The `sbatch` command can take many, many different flags. This is just a quick description of the most popular flags. They are described in more detail further down the page.

Only `-p` and `-t` are required. If you are running a multi-threaded or parallel job, which splits a job in pieces over multiple cores/threads/processors to run faster, `-c or -n` is required.

`-n 1`

Number of tasks requested. By default this value is set to 1 and to each task it is allocated only 1 core. This flag is typically used to request resources when running MPI parallel jobs or other multitasked jobs.

`-c 4`

Number of cores requested per task. Here, we request that the job run on four cores (Some programs use "processors" or "threads" for this idea). For all jobs on O2, you will be restricted to the number of cores you request. If you accidentally request -c 1, but tell your program to use 4 cores, your job will still run but it will be limited to the 1 allocated core only. This is the flag you need to use for the typical parallel job that runs on shared memory systems.

`-e errfile`

Send errors (stderr) to file errfile.

`--mail-user=emailAddress`

Send email to this address. If this option is not used, by default the email is sent to the address listed in `~/.forward`. Must be used with `--mail-type`.

`--mail-type=FAIL`

This option specifies when send job emails. The available mail types include: `NONE`, `BEGIN`, `END`, `FAIL`, `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), and `TIME_LIMIT_50` (reached 50 percent of time limit).

`--mem=100M`

Reserve 100 MB of memory. Use either this option OR `--mem-per-cpu`. To avoid confusion, it's best to include a size unit, one of K, M, G, T for kilobytes, megabytes, etc.

`--mem-per-cpu=100M`

Resource request. Reserve 100 MB of memory **per core**. Use either this option OR `--mem`. Best practice is to include a size unit, such as K, M, G, or T for kilobytes, megabytes, etc.

`--open-mode=append`

For error and output files, choose either to `append` to them or `truncate` them.

`-N 1`

Run on this number of nodes. Unless your job is running in the MPI queue, always use 1.

`-o outfile`

Send screen output (stdout) to file outfile.

`-p short`

Partition to run your job in. All partitions but interactive can be used for `sbatch` jobs.

`-t 30`

Job will be killed if it runs longer than 30 minutes (0-5:30 means 0 days, five hours and thirty minutes)

More `sbatch`

We can specify all the options of `sbatch` in a script and submit. An example batch script is seen below:

myjob.sh

```
#!/bin/bash
#SBATCH -c 1                                # Request one core
#SBATCH -N 1                                # Request one node (if you
request more than one core with -c, also using
             # -N 1 means all cores will be on the same node)
#SBATCH -t 0-00:05                          # Runtime in D-HH:MM format
#SBATCH -p short                            # Partition to run in
#SBATCH --mem=100                           # Memory total in MB (for all
cores)
#SBATCH -o hostname_%j.out                  # File to which STDOUT will be
written, including job ID
#SBATCH -e hostname_%j.err                  # File to which STDERR will be
written, including job ID
#SBATCH --mail-type=FAIL                     # Type of email notification-
BEGIN,END,FAIL,ALL
#SBATCH --mail-user=abc123@hms.harvard.edu   # Email to which notifications
will be sent
hostname
```

The batch script can be submitted like this:

```
sbatch myjob.sh
```

You don't need to use a separate script for submitting jobs with `sbatch`. You can use the `--wrap` option to run a single command. However, we discourage the use of `--wrap` for several reasons: (1) it makes your science less reproducible, (2) it makes jobs harder to troubleshoot (SLURM job accounting doesn't retain commands used with `--wrap`), and (3) certain commands (such as piping, or using `|`) may not be interpreted properly. Instead, we recommend using `sbatch` with a script, or running these commands interactively.

Additionally, if you need some flexibility in your script (e.g. you need to submit the same job but with multiple input files, etc.), submission scripts can take command line arguments as well as inheriting environment variables (your entire current environment is exported with the job when it submits). Here is a shell script that takes a single command line argument:

arguments.sh

```
#!/bin/bash
#SBATCH -c 1                                # Request one core
#SBATCH -N 1                                # Request one node (if you
request more than one core with -c, also using
             # -N 1 means all cores will be on the same node)
#SBATCH -t 0-00:05                          # Runtime in D-HH:MM format
#SBATCH -p short                            # Partition to run in
#SBATCH --mem=100                           # Memory total in MB (for all
cores)
#SBATCH -o hostname_%j.out                  # File to which STDOUT will be
written, including job ID
#SBATCH -e hostname_%j.err                  # File to which STDERR will be
written, including job ID
#SBATCH --mail-type=FAIL                     # Type of email notification-
BEGIN,END,FAIL,ALL
#SBATCH --mail-user=abc123@hms.harvard.edu   # Email to which notifications
will be sent

echo $@ > file.txt
```

If you submit this script like so:

```
sbatch arguments.sh hello world
```

When the job finishes, you should have an output file called `file.txt` that contains the text `hello world`. If you are savvy at bash, you can use this functionality to script loops that can vary the parameters you send to your submission script to submit lots of similar jobs at the same time without having to manually modify the script itself.

You can also take environment variables as well. Say you run the following command:

```
export DESTFILE=file.txt
```

Then make the following change to `arguments.sh` before submitting it as above:

```
echo $@ > $DESTFILE
```

You will still get the same result as above, even though you have not exported `DESTFILE` inside your submission script. The value of `DESTFILE` was exported along with the rest of your environment at submission time, because it was already set. However, for obvious reasons, it is recommended that you make such exports inside your submission script for documentation purposes so that you have a convenient record of what you set those variables to if you return to your submission some time later.

The `srun` command

Like `sbatch`, `srun` can be used to submit jobs under the SLURM scheduler. `sbatch` and `srun` even share many of the same options! However, `srun` is implemented a bit differently. You can use `srun` to create job steps (simply put `srun` in front of the commands you want to run) within an `sbatch` script, or to start an interactive session. If `srun` is used within an `sbatch` script, it will use the preexisting resource allocation. If `srun` is submitted independent of `sbatch`, you will need to specify resources to be allocated to your job.

Interactive Sessions

You can submit "interactive" jobs under SLURM, which allows you to actually log in to a compute node. You can then test commands interactively, watch your code compile, run programs that require user input while the command is running, or use a graphical interface. (Matlab and R can run either in batch mode or graphical mode, for example.) These jobs are intended to be run in the `interactive` partition, but currently can be used in any partition. It may take longer to be assigned an interactive job in a partition other than `interactive`, due to the priority of the partition. See How to choose a partition in O2 for more detail.

```
[kmk34@login05 ~]$ srun --pty -p interactive --mem 500M -t 0-06:00
/bin/bash
[kmk34@compute-a-16-68 ~]$
```

You can then run your commands, and logout when you're done. The `interactive` queue has a limit of 12 hours.

You can request extra memory or multiple cores (up to 20) in an interactive session `srun`.

Note: It is not recommended to submit `srun` commands from within an interactive job. These commands will be executed on the same resource allocation as the interactive job. If you want to submit other jobs from within an interactive job, use `sbatch` commands instead.

Within `sbatch` scripts

With the SLURM scheduler, it is encouraged to put `srun` in front of every command you want to run in an `sbatch` script. Using `srun` will denote job steps, so when you monitor your job, each `srun` command will be shown as a different step. Additionally, if your job fails, job steps will help you troubleshoot better by narrowing down which command had the issue.

Here is an example script that will give the hostname of the node it was executed on, and print a message.

```
#!/bin/bash
#SBATCH -c 2                                # 1 core
#SBATCH -t 0-00:05                          # Runtime of 5 minutes, in
D-HH:MM format
#SBATCH -p short                            # Run in short partition
#SBATCH -o hostname_%j.out                  # File to which STDOUT + STDERR
will be written, including job ID in filename
#SBATCH --mail-type=FAIL                     # ALL email notification type
#SBATCH --mail-user=abc123@hms.harvard.edu  # Email to which notifications
will be sent

srun -c 1 hostname
srun -c 1 echo "Hello, I ran an sbatch script with srun commands!"
```

If we save this script as `srun_in_sbatch.sh`, it can be submitted by `sbatch srun_in_sbatch.sh`. After the job completes, you can see the job statistics (which will be broken down by numbered job steps) by running `sacct -j <jobid>`.

Monitoring Jobs

There are several commands you may wish to use to see the status of your jobs, including:

- `squeue` - by default `squeue` will show information about **all** users' jobs. Use `-u <userid>` to get information just about yours.
- `scontrol` - most `scontrol` options can't be invoked by regular users, but `scontrol show job <jobid>` is a useful command that gives detailed job information. This command only works for currently running jobs.
- `sstat` - shows status information for currently running jobs. Many fields can be requested using the `--format` parameter. Reference the job status fields in the sstat documentation for more information. Unless you are running a more experienced user running a job with multiple steps, you probably want to add `.batch` to the jobid of the job.
- `sacct` - reports accounting information for jobs and job steps. This works for both running or completed jobs, but it is most useful for

completed jobs. Many fields can be requested using the `--format` parameter. Check the job accounting fields in the sacct documentation for more information.

Example monitoring commands

```
# list information about all non-completed jobs for a user, including job
ids and what status they're in.
squeue -u <userid>

# list information for a single job
squeue -j <jobid>

# list information for only running jobs
squeue -t RUNNING

# list information only for pending jobs
squeue -t PENDING

# list information for only jobs in short partition
squeue -p short

# list information for jobs in short partition that are currently running
for a user
squeue -p short -u <userid> -t RUNNING

# show status information for running job
# you can find all the fields you can specify with the --format parameter
by running sstat -e
# Advanced users may want to use -j <jobid>.1 for step 1 of a job
sstat -j <jobid>.batch --format JobID,MaxRSS,MaxVMSize,NTasks

# show details for a running job
# -dd requests more detail
scontrol show job <jobid> -dd

# get statistics on a completed job
# you can find all the fields you can specify with the --format parameter
by running sacct -e
# you can specify the width of a field with % and a number, for example
--format=JobID%15 for 15 characters
sacct -j <jobid>
--format=JobId,AllocCPUs,State,ReqMem,MaxRSS,Elapsed,TimeLimit,CPUTime,Req
Tres
```

Controlling jobs

There are several ways to modify your submitted jobs in SLURM, such as:

- canceling a job using `scancel`
- pause a job using `scontrol`
- release a job using `scontrol`

Example job monitoring commands

```
# Cancel a job.
# The scancel command can also be used to kill job arrays or job steps.
scancel <jobid>

# Cancel all your jobs
scancel -u <userid>

# Cancel all of your running jobs
scancel -t RUNNING -u <userid>

# cancel a job using the name given during submission, not using job id
scancel --name BlastJob

# Prevent a queued job from starting
scontrol hold <jobid>

# Allow a held job be scheduled again
scontrol release <jobid>
```

Job Completion

By default, you will not receive an execution report by e-mail when a job you have submitted to Slurm completes. If you would like to receive such notifications, please use `--mail-type` and optionally `--mail-user` in your job submission. Currently, the SLURM emails contain minimal information (jobid, job name, run time, status, and exit code); this was an intentional design feature from the Slurm developers. We are looking into alternative methods to deliver job metrics, such as requested number of nodes and cores, and used memory. At this time, we suggest running `sacct` queries for more detailed information.

Troubleshooting jobs

The emailed job reports will be useful to find out the exit code, status, and run time of your job. If a job fails, the subject line will (usually) say `FAILED` or `CANCELLED` instead of `COMPLETED`. More information for a job can be found by running `sacct -j <jobid>`. The following command can be used to obtain accounting information for a completed job:

```
sacct -j <jobid> --format
JobId,NNodes,Partition,NCPUs,State,ReqMem,MaxRSS,Elapsed,CPUTime,TimeLimit
,ExitCode,Start,End
```

If you get a `TIMEOUT` error, the `CPUTime` reported will be greater than the `-t` limit you specified in your job.

Similarly, if your job used too much memory, you will receive an error like: `Job <jobid> exceeded memory limit <memorylimit>, being killed`. For this job, `sacct` will report a larger `MaxRSS` than `ReqMem`, and `CANCELLED` job status. You will need to rerun the job, requesting more memory.

For much more information on figuring out why jobs don't start, or don't finish, see the separate page on troubleshooting Slurm jobs.

If you are contacting Research Computing because a job did not behave as expected, it's often helpful to include the job report in your request. Just attach it to the email, or paste it into the form.

Job output handling

By default, your job output will not be emailed to you, even if you request email notifications. Slurm will automatically put job error and output in a file called `slurm-<jobid>.out` in the directory you submitted the job from.

If you would prefer, you can direct the output by specifying the `-o` (stdout) and `-e` (stderr) options to `sbatch`:

```
sbatch -o myjob.out -e myjob.err myjob.sh
```

You can add the execution node name and jobids to the output and error files using `%N` and `%j`:

```
sbatch -o %N_%j.out -e %N_%j.err myjob.sh
```

Running graphical programs on O2 with X11

A number of programs with graphical user interfaces (e.g., R, Matlab) use the X11 system which lets the program run on an O2 computer, but show the graphics on your desktop. To do this, you need to have an X11 server running on your desktop, and your SSH connection needs to have X11 forwarding enabled. See this page for X11 setup instructions.

You will need to connect to O2 with the `-XY` flags in your SSH command. Substitute <username> for your eCommons:

```
$ ssh -XY <username>@o2.hms.harvard.edu
```

To enable graphics forwarding within an `srun` job, add the `--x11` flag to your job submission. For example:

```
$ srun --pty -p interactive -t 0-0:5 --x11 bash
```

Likewise, add the `--x11=batch` flag for graphics forwarding within `sbatch` job submissions.

Running jobs on multiple cores

Modern computers have multiple cores, allowing them to run multiple jobs at once. Many programs allow you to use multiple cores for a single run. (Just to confuse you, different programs talk about running on multiple "processors" or "cores" or "threads" or "CPUs".) This can allow you a substantial speedup, but there are some important issues to be aware of.

Unlike multi-core programs, truly parallel programs use a system like MPI to let you run on more than one computer at a time. For information on that, see the Parallel Jobs on O2 page.

How many cores?

First, you need to tell SLURM the number of cores you want to run on. You do this with the -c flag. The number of cores you request from the program should always be the same as the number of cores you request from Slurm. Note that different programs have different options to specify multiple cores. For example, `tophat -p 8` asks the Tophat aligner for eight cores. So you might have a job submission script that has a line to request 8 cores:

```
#SBATCH -c 8
```

Later on in the script, you have a tophat alignment command, which also specifies using 8 cores:

```
tophat -p 8 ...
```

If you accidentally ask for only one core in the job submission (`sbatch -c 1`), but try to let your program use multiple cores (`tophat -p 8`), you will not be able to do so. On O2, CPU usage is restricted to the cores you request, which is performed by the Cgroups plugin.You will not receive an explicit error message, but nevertheless, your job will be confined to the allocated cores. For this reason, you may observe a performance decay when running jobs on O2 instead of Orchestra, because in Orchestra your jobs were using multiple cores that were not allocated to you.

Time limits

Each job has a runtime limit, which is the maximum number of seconds that the job can be in RUN state. This is also known as "wall clock time". You specify this limit with the `-t` parameter with `sbatch` or `srun`. If your job does not complete prior to the runtime limit, then your job will be killed. Each partition has a default runtime limit, and a maximum time limit. See below for details:

Queue	Default time limit	Maximum time limit
short	2 hours	12 hours
medium	1 day	5 days
long	1 day	30 days
interactive	none	12 hours
mpi	1 day	5 days
priority	1 day	30 days
transfer	1 hour	5 days
gpu	1 hour	see Using O2 GPU resources

Requesting resources

You may want to request a node with specific resources for your job. For example, your job may require 4 GB of free memory in order to run. Or you might want one of the nodes set aside for file transfers or other purposes.

Memory requirements

Every job requires a certain amount of memory (RAM, "active" memory, not disk storage) to run. If the jobs on a given node use too much memory, this memory exhaustion can adversely impact other running jobs, prevent the dispatch of new jobs, cause nodes to crash, and unfairly benefit a few intensive users at the expense of all other users of that node.

Jobs that use excessive memory without requesting resources may be terminated by Research Computing to allow other jobs to run.

If your job requires more than 1 GB of memory, please override the defaults. For example, to submit a job to the short queue and reserve 8 GB of memory for the duration of the job:

```
sbatch -p short -t 8:00 --mem=8000 your_job.sh
```

Note that without any units appended, memory reservation values are specified in MB by default. You can change the size unit of the memory request by appending K, G, T for kilobytes, gigabytes, terabytes etc.

Filesystem Resources

Filesystem resources make sure that your job is only dispatched to a node that has the appropriate network file system mounted and accessible to it. All users are encouraged to use filesystem resources. During planned maintenance events or unplanned filesystem outages, using a filesystem resource requirement will keep your job from being unnecessarily dispatched and subsequently failing.

The file system resources are:

- `groups` (`/n/groups`)
- `log` (`/n/log` - for web hosting users)
- `files` (`/n/files` - accessible through `transfer` partition, but you must request access to run jobs in this partition)
- `scratch3` (`/n/scratch3` - for storage of temporary or intermediary files)

Example Usage:

```
sbatch --constraint="files"
```

If your job requires multiple filesystems:

```
sbatch --constraint="files&groups"
```

Feedback Wanted

We expect to make adjustments to the SLURM configuration as we learn more about how O2 is used. Your feedback will be the best way for us to learn what adjustments we should be considering to make the cluster most useful to the widest audience. We're also interested in feedback on what information is most useful for new users, so that we can expand and improve this documentation. Please give us feedback early and often. Please use the support form on the Research Computing web site.