Lab 6: SCF and Parallel Computing

2024-10-11

Table of contents

Overview

Today we'll discuss the Statistical Computing Facility's (SCF) cluster, which is administrated by the Department of Statistics. Much of this material comes directly from the SCF's documentation; see the links below for more details:

- SCF homepage
- SCF Linux cluster
- SCF FAQs and How-Tos
- SCF Parallel processing in Python tutorial
- How to get help on the SCF

- SCF JupyterHub

SCF cluster capabilities and hardware

Clusters, partitions, and nodes! Oh my!

A **cluster** is a collection of computing **nodes**. A node is analogous to a laptop or desktop computer, with an operating system, RAM, CPUs, and sometimes GPUs.

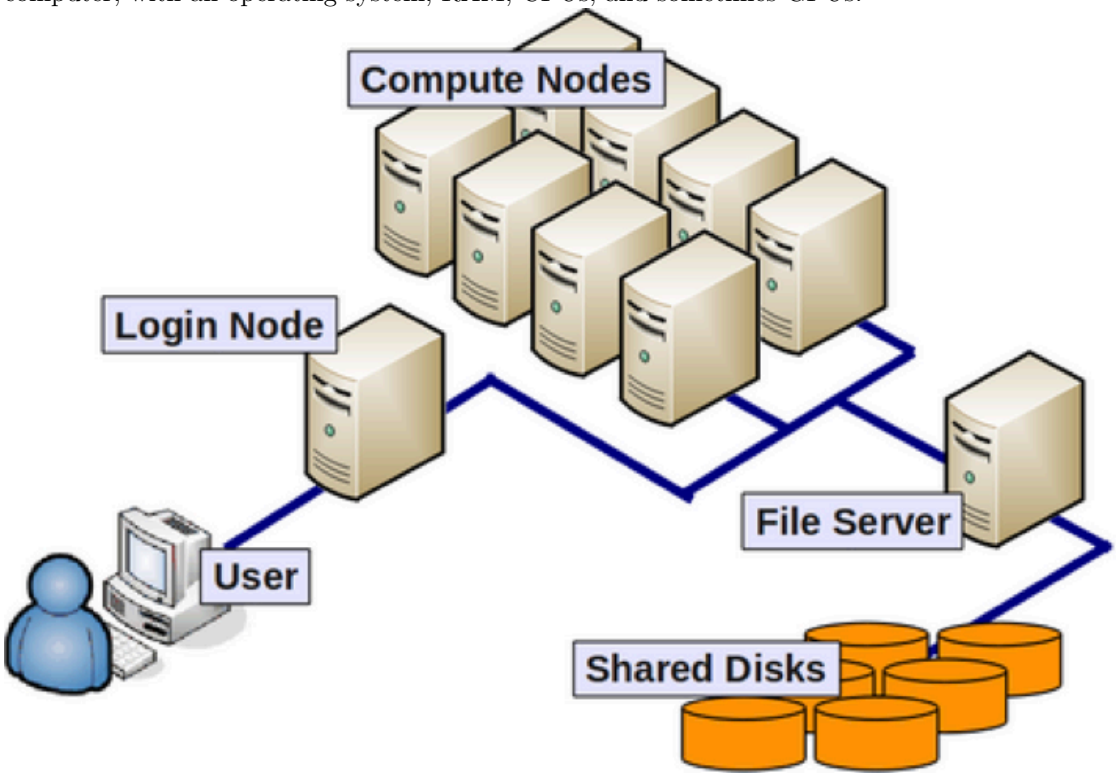

Image source

Nodes with similar capabilities or access policies are often grouped together as a **partition**. This allows cluster users to request the resources they need – such as number of CPU **cores** – from a pool of nodes, rather than specify a specific one.

The SCF cluster has 1064 cores, across 7 partitions with more than 26 nodes, but for this class we'll use the `low` partition. It's the default, so you won't need to do anything special to use it for your jobs (we'll explain how to submit jobs later on).

The `low` partition has the following resources:

- 8 nodes
- 32 cores per node

- 256 GB RAM per node

The SCF Linux cluster uses a batch job scheduler called Slurm, commonly used by large computer centers in both academia and industry. Although the SCF cluster has much greater computational capacity than our personal computers, it is a shared resource with many users. With Slurm, the cluster is able to schedule jobs asynchronously, balancing resource allocations so that everyone gets a turn. This also means that when you submit a job, it may not run until many minutes or hours later. Be sure to give yourself enough time!

Your ~/ on the SCF

Nodes in the SCF cluster use the Linux distribution Ubuntu. Every user has a private home directory which, as usual on Linux-based OSes, has the shortcut `~/`. In this section, we'll give an overview of access, disk space, and remote file transfers to and from your SCF home directory.

Logging in

The SCF has a number of **login nodes** which you can access via `ssh`.

> **Note**
>
> For info on using `ssh` (including on Windows), see here.

For example, I'll use `ssh` to connect to the `gandalf` node:

```
james@pop-os:~$ ssh <scf-username>@gandalf.berkeley.edu
The authenticity of host 'gandalf.berkeley.edu (128.32.135.47)' can't be established.
ED25519 key fingerprint is SHA256:io5uUQGbCZie78mF+UUZ5guDK29JXQGQ6LVB129UoUo.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'gandalf.berkeley.edu' (ED25519) to the list of known hosts.
```

Notice that upon first connecting to a server you haven't visited there is a warning that the "authenticity of the host … can't be established". So long as you have typed in the hostname correctly (`gandalf.berkeley.edu`, in this case), and trust the host (we trust the SCF!) then you can type `yes` to add the host to your known hosts file (found on your local machine at `~/.ssh/known_hosts`).

You'll then be asked to enter your password for the SCF cluster. For privacy, you won't see anything happen in your terminal when you type it in, so type carefully (you can use `Backspace` if you make a mistake) and press `Enter` when you're done. If you were successful, you should see a welcome message and your shell prompt, like:

```
<scf-username>@gandalf:~$
```

To get your bearings, you can type `pwd` to see where your home directory is located on the SCF cluster filesystem:

```
<scf-username>@gandalf:~$ pwd
/accounts/grad/<scf-username>
```

Your home directory is likely also in the `/accounts/grad/` directory, as mine is.

Other login nodes

> **!** Important
>
> Don't run computationally intensive tasks on the login nodes!
> They are shared by all the SCF users, and should only be used for non-intensive interactive work such as job submission and monitoring, basic compilation, managing your disk space, and transferring data to/from the server.

If for some reason `gandalf` is not working for you, the SCF has a number of nodes which can be accessed from your local machine with commands of the form `ssh <scf-username>@<hostname>.berkeley.edu`. Currently, these are:

- `aragorn`
- `arwen`
- `dorothy`
- `gandalf`
- `gollum`
- `hermione`
- `quidditch`
- `radagast`
- `shelob`

Disk space

Your home directory has a limited amount of disk space; you can check how much you have used and available using the `quota` command, which will show your home directory usage and quota on the line for the `accounts` filesystem.

```
<scf-username>@gandalf:~$ quota
FILESYSTEM  USE         QUOTA       %
  accounts  5.29 G      20 G        26.5
```

The `accounts` filesystem is accessible from all the nodes on the SCF cluster. This means that regardless of which login or compute node you use, you will have access to the files in your home directory. Moreover, your home directory is backed up regularly, so you are protected from accidental data loss.

If you're running out of space, you should try to selectively delete large files that you no longer need. See here for some tips on finding large files. If all else fails, you can request additional space or request access the `/scratch` filesystem. The latter is a good place for large datasets, but note that `/scratch` is not backed up, unlike your home directory. See the previous link for more info. Note though that

disk space should not be a concern for Stat 243, but being aware of `/scratch` can be relevant for Stat graduate students (MA and PhD) working on projects in the future.

For temporary files (e.g., intermediate results of a computation that you don't need to store for later), every machine has a `/tmp` filesystem. However, `/tmp` is *always linked to the specific machine* you are on, meaning that if you put something in `/tmp` on `gandalf` and then later go to `aragorn`, you won't find your files in the `/tmp` directory there. If you go back to `gandalf`, you will likely find them again, but `/tmp` is automatically wiped when a machine reboots, so only use it for files you don't care about preserving!

Data transfer: SCP / SFTP

We can use the `scp` and `sftp` protocols to transfer files to and from any login node on the SCF cluster. `scp` is a shell program that should be available by default on macOS and Linux, while on Windows you can use WinSCP. WinSCP can also do `sftp` transfers, or you can use FileZilla on any platform for `sftp`. Both WinSCP and FileZilla have a GUI that allows you to drag and drop files between your local machine and a remote host.

The syntax for `scp` is `scp <from-place> <to-place>`, and you'll typically use `scp` on your local machine. For example, we'll show how to transfer the file `data.csv` (you can find it here) :

To SCF while on your local machine

```
# transfer to my home directory without renaming the file
scp data.csv <scf-username>@gandalf.berkeley.edu:~/

# transfer to the data/ subdirectory and rename the file
scp data.csv <scf-username>@gandalf.berkeley.edu:~/data/new_name.csv

# transfer to the gandalf-specific /tmp/ directory
scp data.csv <scf-username>@gandalf.berkeley.edu:/tmp/
```

From SCF while on your local machine

```
# now <from-place> is a path in my home directory on the SCF
scp <scf-username>@gandalf.berkeley.edu:~/data/new_name.csv ~/Desktop/data_scf.csv
```

For more information, see here. In particular, if you have a very large dataset to transfer, Globus is a better option than either `scp` or `sftp`.

Running jobs on the SCF

> Tip
>
> For this section, you'll need to have some of this week's materials in your home directory on the SCF cluster.
> To get them, you can (and probably should) clone the course repo to your home directory on the SCF cluster:

```
# first ssh to an SCF login node: ssh <scf-username>@<host-name>.berkeley.edu

cd ~/ # or a subdirectory if you prefer

git clone https://github.com/berkeley-stat243/fall-2024.git
# or git@github.com:berkeley-stat243/fall-2024.git if you use SSH with git

# the files you need are here
cd fall-2024/labs/lab6-scfparallel/
```

As mentioned before, the SCF cluster uses Slurm to schedule jobs. We'll learn some Slurm commands now.

Non-interactive jobs: `sbatch`

> Tip
>
> This is the best option when you have a long-running job where no

The first command is **`sbatch`**. To use **`sbatch`**, you will first create a bash script (you can do this on the SCF cluster if you know how to use a shell text editor like **`nano`**, or just do it locally and transfer the file to the SCF via one of the methods discussed before).

In the course repo, you will find an example bash script called **`submit.sh`**. Here's what it contains:

```
#!/bin/bash

#################
# SBATCH OPTIONS
#################

#SBATCH --job-name=example # job name for queue (optional)
#SBATCH --partition=low    # partition (optional, default=low)
#SBATCH --error=ex.err     # file for stderr (optional)
#SBATCH --output=ex.out    # file for stdout (optional)
#SBATCH --time=00:01:00    # max runtime of job hours:minutes:seconds
#SBATCH --nodes=1          # use 1 node
```

```
#SBATCH --ntasks=1          # use 1 task
#SBATCH --cpus-per-task=1   # use 1 CPU core

####################
# Command(s) to run
####################

./example.sh
```

The `example.sh` script creates a file called `testFile.txt` and appends some output to that file. Note that since `submit.sh` is itself a bash script, you could have just put the contents of `example.sh` directly in `submit.sh`, but it is good practice to keep your computational code separate from your Slurm batch submission scripts.

Now we can submit the job using the Slurm command `sbatch`:

```
sbatch submit.sh
```

Once you run that, your job will be queued. To check it's status, you can use `squeue`:

```
squeue -u <scf-username> # use your username, not mine
```

And now you wait. Go get a cup of coffee or get some rest! Then check back later.

Interactive jobs

> **Tip**
>
> Interactive jobs are the best option when you need to do many short but computationally intensive tasks which require your hands at the keyboard.

`srun`

> **Tip**
>
> This is the best option when you want to work at the Linux command line and don't need a GUI, e.g. when debugging components of larger non-interactive job.

To run an interactive job via Slurm, you can use the `srun` command:

```
# ssh -Y <scf-username>@gandalf.berkeley.edu

# 10 minute time limit
srun  -t 00:10:00 -N 1 -n 1 -c 1 --pty /bin/bash

# uses resources already made available from the previous srun command
srun --pty --x11=first matlab # interactive matlab job
```

Here we used the -Y flag to `ssh` to run software with a GUI, such as MATLAB, which allows the GUI to open in our local machine while still running computations on the SCF cluster. For this to work, you'll need X server software on your own machine to manage the graphical windows. For Windows, your options include eXceed or Xming and for Mac, there is XQuartz.

JupyterHub for RStudio or Jupyter Notebooks

> **Tip**
>
> This is the best option when you want to interact with the RStudio or Jupyter GUI while doing your computations or debugging.

The SCF's JupyterHub is another resource for interactive computing on the SCF, allowing you to use RStudio or Jupyter on one of the cluster's computing nodes.

Specifying resources

If you are submitting a job that uses multiple cores or nodes, you may need to carefully specify the resources you need. The main key flag for use in your job script is:

- `--ntasks-per-node`: indicates the number of tasks (i.e., processes) one wants to run on each node

The value passed to `--ntasks-per-node` becomes available through the `SLURM_NTASKS` environment variable, which can be retrieved in python via:

```
import os
ntasks = os.getenv("SLURM_NTASKS")
```

Specifying resources: alternatives

The `--ntasks-per-node` flag is convenient as it does not require the user to know the actual number of CPUs on each node. It is generally a good choice if one is simply running parallel code via multiple processes on one node. However, there are alternatives flags that can be used in conjunction for finer-grained control:

- `--nodes` (or `-N`): indicates the number of nodes to use
- `--ntasks` (or `-n`): indicates the number of tasks to run, not necessarily on the same node
- `--cpus-per-task` (or `-c`): indicates the number of CPUs to be used for each task

Since specifying `--ntasks` does not guarantee that all the cores will be on a single node, this can cause problems in the common case of code that works on one node but not multiple nodes. If one wants to do parallelization at multiple levels (e.g., multiple processes, each process using multiple threads, such as for linear algebra), then one would use both `--ntasks-per-node` and `--cpus-per-task`. In general `--cpus-per-task` will be 1 except when running threaded code.

The `SLURM_NTASKS` environment variable is set by Slurm when the job starts running, and therefore can be accessed within your jobs. In addition to `SLURM_NTASKS` here are some other variables that

may be useful: `SLURM_CPUS_ON_NODE`, `SLURM_CPUS_PER_TASK`, `SLURM_NODELIST`, `SLURM_NNODES`. An explanation of each one of those can be found on slurm's manual (`man sbatch`).

Monitoring your jobs

As we saw, the basic command for seeing what is running on the system is `squeue`:

```
squeue # this shows all the queued jobs!
squeue -u SCF_USERNAME # this shows your jobs
```

To see what nodes are available in a given partition, you can use `sinfo`:

```
sinfo -p low
```

Finally, you can cancel a job with `scancel`.

```
# you can find your job ID using `squeue`
scancel YOUR_JOB_ID
```

See the "How to Monitor Jobs" section of the SCF's documentation on this page for some more info.

Exercise

You can either do 1. on your local machine and transfer the file to the SCF cluster, or you can do it remotely. 2 - 4 should be done on the SCF.

1. Create a submission `submitBoot.sh` that will execute the code in `boot_proc.py`. To do this I suggest copying the text from `submit_py.sh` and updating a couple lines:

- Change `job-name` to whatever you want to call this job.
- Change time to `00:10:00` just in case your code runs longer than one minute.
- Change the `--cpus-per-task` to 3. This is where you are telling Slurm to execute in parallel on 3 CPUs.
- Change the last line `python boot_serial.py`.

2. Call `sbatch submitBoot.sh` to execute the python script `boot_proc.py`.
3. Check that the execution worked by examining the `py.out` file.
4. Note that the number of Dask workers in `boot_proc.py` and `boot_dist.py` is hard-coded. In many cases, it would be better for this value to be set dynamically based on an appropriate Slurm environment variable. Make this small adjustment to those files and verify it works as intended.

Acknowledgements

This lab was developed by Andrew Vaughn and James Duncan for R, adapted to python by Ahmed Eldeeb and slightly adjusted for the Fall 2024 semester.