SCF Computing Cluster

Andrew Vaughn

18 October, 2021

Introduction

Much of this material comes from the SCF documentation at the SCF homepage.

Outline

This training session will cover the following topics:

- System capabilities and hardware
  - SCF computing nodes
  - Disk space
- Logging in, data transfer, and software
  - Logging in
  - Data transfer: SCP/SFTP
  - Software modules
- Submitting and monitoring jobs
  - Accounts and partitions
  - Interactive jobs
  - Basic job submission
  - Parallel jobs
  - Monitoring jobs and cluster status
- Exercise

System Capabilities and Hardware

- The SCF Linux cluster contains 352 cores, separated into 12 nodes and two partitions.
  - *Low* partition
    - 8 nodes
    - 256 total cores
    - each node contains 32 cores and 256 GB of RAM
  - *high* partition
    - 4 nodes
    - 96 total cores

    - each node contains 24 cores and 128 GB of RAM
  - *GPU* partition
    - Tesla K20Xm
    - one node in the high partition
    - 2 CPU cores and 128 GB of RAM (shared with CPU)

SCF Computing Nodes

The SCF cluster is has several compute nodes, which you can list using `sitehosts compute`.
They are also listed on this page.

The nodes are divided into 3 partitions, listed above, with restrictions associated with each. Any job you submit must be submitted to a partition to which you have access.

For the puposes of this class, we only have access to the *low* partition.

Disk Space

Every account is given 5 GB of disk space. (This can be queried using `quota <account_name>`) This includes your home directory and a `/tmp` directory tied to your account.
Space in the `/scratch` partition is available by request only.

Logging in

You can use login nodes for non-intensive interactive work such as job submission and monitoring, basic compilation, managing your disk space, and transferring data to/from the server.

To login, you need to have software on your own machine that gives you access to a UNIX terminal (command-line) session. These come built-in with Mac (see `Applications -> Utilities -> Terminal`). For Windows, some options include PuTTY.

Here are instructions for doing this setup, and for logging in.

Login servers include: `arwen, beren, bilbo, gandalf, gimli, legolas, pooh, radagast, roo, shelob, springer, treebeard`

Then to login:

```
ssh SCF_USERNAME@LOGINSERVER.berkeley.edu
```

Then enter your password.
One can then navigate around and get information using standard UNIX commands such as `ls`, `cd`, `du`, `df`, etc.

Data transfer: SCP/SFTP

We can use the *scp* and *sftp* protocols to transfer files from any login node on the SCF cluster.

For example, we show how to transfer the file `data.csv`.

To transfer that to your directoy on the SCF cluster, the syntax for `scp` is `scp <from-place> <to-place>`

Below are some examples of transferring this file to various locations on the remote machine (your home directory, a folder called data in your home directory, and the tmp folder)

```
# to SCF, while on your local machine
scp data.csv andrew_vaughn@arwen.berkeley.edu:~/
scp data.csv andrew_vaughn@arwen.berkeley.edu:~/data/newName.csv
scp data.csv andrew_vaughn@arwen.berkeley.edu:/tmp/
```

Here is how to transfer data from the SCF to your local machine, while on your local machine. This will transfer the `new_name.csv` file to your Desktop with the name `data_scf.csv`

```
# from SCF, while on your local machine
scp andrew_vaughn@arwen.berkeley.edu:~/data/new_name.csv ~/Desktop/data_scf.csv
```

One program you can use with Windows is *WinSCP*, and a multi-platform program for doing transfers via SFTP is *FileZilla*. After logging in, you'll see windows for the SCF filesystem and your local filesystem on your machine. You can drag files back and forth.

Submitting jobs: accounts and partitions

All computations are done by submitting jobs to the scheduling software that manages jobs on the cluster, called SLURM (Simple Linux Utility for Resource Management).

There is nothing you must specify to submit jobs to the SCF cluster, however, there are several options that are good form (more on this below).

Interactive jobs

You can do work interactively.

For this, you may want to have used the -Y flag to ssh if you are running software with a GUI such as MATLAB.

```
# ssh -Y SCF_USERNAME@arwen.berkeley.edu
srun --pty /bin/bash           # interactive bash job
srun --pty --x11=first matlab  # interactive matlab job
```

To display the graphical windows on your local machine, 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*.

Submitting a batch job

Or you can submit a job to run in the background.

Let's see how to submit a simple job. If your job will only use the resources on a single node, here's an example job submission script, *exSub* in the section folder:

```
#!/bin/bash

####################
# SBATCH OPTIONS
####################
#SBATCH --job-name=example              # job name fore queue, default may be u$
#SBATCH --partition=low                 # high/low/gpu, default if empty is low
#SBATCH --error=ex.err                  # error file, default if empty is slurm$
#SBATCH --output=ex.out                 # standard out file, no default
#SBATCH --time=00:01:00                 # optional, max runtime of job hours:minutes:seconds
#SBATCH --nodes=1                       # only use 1 node, MPI option
#SBATCH --ntasks=1                      # how many tasks to start
#SBATCH --cpus-per-task=1               # number of cores to use, multi-core/mu$

####################
# What to run
####################

./example.sh
```

This script runs the following bash script, *example.sh* in the section folder:

```
#!/bin/bash

# set file name
fileName="testFile.txt"

# create file
touch $fileName

# intro message
echo "I'm your new test script!" >> $fileName
echo >> $fileName
echo >> $fileName

# fill it with some things
for i in {1..10}
do
        echo $i >> $fileName
done

# finish it out
echo >> $fileName
echo >> $fileName
echo "All Finished!" >> $fileName
```

Now let's submit and monitor the job:

```
sbatch exSub

squeue -u <SCF_USERNAME>
```

Parallel job submission

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

- `--nodes` (or `-N`): indicates the number of nodes to use
- `--ntasks-per-node`: indicates the number of tasks (i.e., processes) one wants to run on each node
- `--cpus-per-task` (or `-c`): indicates the number of cpus to be used for each task

In addition, in some cases it can make sense to use the `--ntasks` (or `-n`) option to indicate the total number of tasks and let the scheduler determine how many nodes and tasks per node are needed. In general `--cpus-per-task` will be 1 except when running threaded code.

When setting up parallel R code, you can find out how many cores there are on the node assigned to you with:

```
ncores <- Sys.getenv("SLURM_CPUS_ON_NODE")
```

In addition to SLURM_CPUS_ON_NODE here are some of the variables that may be useful: SLURM_NTASKS, SLURM_CPUS_PER_TASK, SLURM_NODELIST, SLURM_NNODES.

Monitoring jobs and the job queue

The basic command for seeing what is running on the system is `squeue`:

```
squeue
squeue -u SCF_USERNAME
```

To see what nodes are available in a given partition:

```
sinfo -p low
sinfo -p high
sinfo -p gpu
```

You can cancel a job with `scancel`.

```
scancel YOUR_JOB_ID
```

The SCF has some tips about monitoring your job.

Exercise

1) Create a submission `bootSub` that will execute the code in `boot.R`. To do this I suggest copying the text from `exSubR` and updating a couple lines:

    - Change job-name to whatever you want to call this job
    - Change rEx.err and rEx.out to boot.err and boot.out. Note these files can be useful for fixing bugs, particuarily when excecuting shell scripts. These files show errors and output printed to the terminal.

- 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 nodes.
- Change the last line `R CMD BATCH --no-save boot.R boot.Rout`. The `boot.Rout` file will contain all of code, messages, errors, etc. from the R session.

2) Transfer the files boot.R and bootSub to your SCF account. This can be done using scp before logging onto the SCF cluster (see examples above)

    - Note, there are other ways to transfer files. For example, one other option would be to clone the repo for this class onto your SCF account after logging in.

3) Call `sbatch bootSub` to excecute the R script `boot.R`.

4) Check that the excecution worked by examining the `.Rout` file.

5) If you have time:

    - Update parallel code to be execute with the `future` package to compare the results.