SCF Computing Cluster

Jared Bennett

21 October, 2019

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 pupuses 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, the file `bayArea.csv` is too large to store on Github; you can obtain it here.

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

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

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

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.

Software modules

A lot of software is available on SCF but needs to be loaded from the relevant software module before you can use it.

```
module list  # what's loaded?
module avail  # what's available
```

One thing that tricks people is that the modules are arranged in a hierarchical (nested) fashion, so you only see some of the modules as being available *after* you load the parent module. Here's how we see the Python packages that are available.

```
which R # R is /usr/bin/R


module avail
module load R/3.5.1
which R # R is /usr/local/linux/R-3.5.1/bin/R
module avail
```

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).

You can see what accounts you have access to and which partitions within those accounts as follows:

```
sacctmgr -p show associations user=SCF_USERNAME
```

Here's an example of the output for me. I have access to 2 accounts.

```
Cluster|Account|User|Partition|Share|GrpJobs|GrpTRES|GrpSubmit|GrpWall|GrpTRESMins|MaxJobs|MaxTRES|MaxTI
screms|site|jared_bennett||1||||||||||||normal|||
sm|site|jared_bennett||1||||||||||||normal|||
```

To see what groups your user is a part of, you can use `groups SCF_USERNAME`.

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:

```bash
#!/bin/bash

####################
# SBATCH OPTIONS
####################
#SBATCH --job-name=example            # job name fore queue, default may be user-id, but unsure
#SBATCH --partition=low               # high/low/gpu, default if empty is low
#SBATCH --error=/accounts/campus/jared_bennett/ex.err   # error file, default if empty is slurm-<jobid>
#SBATCH --output=/accounts/campus/jared_bennett/ex.out  # standard out file, no default
#SBATCH --time=00:01:00               # 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/multi-thread jobs


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

/accounts/campus/jared_bennett/example.sh
```

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

```bash
#!/bin/bash

# set file name
fileName="/accounts/campus/jared_bennett/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..50}
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 -j <JOB_ID>
```

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
squeue -A s243
```

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
```

For more information on cores, QoS, and additional (e.g., GPU) resources, here's some syntax:

```
squeue -o "%.7i %.12P %.20j %.8u %.2t %.9M %.5C %.8r %.3D %.20R %.8p %.20q %b"
squeue -p low,high,gpu -o "%.14i %.6P %.20j %.12u %.2t %.11l %.11M %.11V %.5C \
%.8r %.6D %.20R %.13p %8q %b"
```

We provide some tips about monitoring your job.
```

# Exercise

Consider the Wikipedia traffic data in `/scratch/users/paciorek/wikistats/dated_2017_small/dated` on the SCF cluster.

1) In an interactive shell, copy the first 30 files to your `/tmp` directory. (recall the use of wildcards in file globbing to select a set of files, from the unit/tutorial on bash)

2) Run R code to do the following: Using the future package, with either future_lapply or foreach with the doFuture backend, write code that, in parallel, reads in the space-delimited file and filters to only the rows that refer to pages where "Barack_Obama" appears. You can use the code from unit7-parallel.R as a template, in particular the chunks labeled 'rf-example', 'foreach' and 'future_lapply'. Collect all the results into a single data frame. Run your code using an interactive session.

3) Now replicate what you did above but using sbatch to submit your job as a batch job, where step 2 involves running R from the command line using R CMD BATCH.

Note that as we saw in class, the data are the number of hits on different Wikipedia pages for November 4, 2008. The columns are: date, time, language, webpage, number of hits, and page size.