Faster Computing - Part 2

December 3, 2020

Contents

Today we'll be interacting with Cosine's *RStudio Server Pro* and with their *High Performance Cluster* (HPC). In order to access either of these from off-campus, you'll need to be connected to OSU's Virtual Private Network (VPN). OSU uses the *CISCO AnyConnect* client, so if you haven't already installed it, do so now. See download and instructions at https://oregonstate.teamdynamix.com/TDClient/1935/Portal/KB/ArticleDet?ID=76790. You'll need to confirm your access privileges using *Duo* and your ONID credentials.

In order to move files between your computer and the servers, you'll also need an `sftp` client, like https://cyberduck.io.[1]

Finally, in order to pass commands to the HPC, you'll need an `ssh` client. Mac users can use *Terminal*, which is built in to the operating system. Windows users will need to install one (e.g., https://mobaxterm.mobatek.net, which also does `sftp`).

RStudio Server Pro

This one's super simple in that all you have to do is use your browser and ONID credentials to login to

https://rstudio-1.cosine.oregonstate.edu.

If nothing happens, make sure you're logged into the VPN first. Once you're in, you'll be presented with a screen very much like the standard *RStudio* interface with your "*R* session" already initiated.[2] You can start typing commands into the console just as you would on your computer. Of course, you're unlikely to want to do that as you've probably got your code all written and tested on smaller datasets/simulation runs (i.e. you are using the server not for writing or debugging, but just to run code).

To copy files between your computer and the server, `sftp` into the server. In *Cyberduck*:

1. click on "*Open Connection*",
2. select "*SFTP (Secure File Transfer Protocol)*",

[1] File transfer can also be done by `ssh` command-line, but there are no commands to remember with `sftp`.

[2] If you wish to initiate additional sessions (e.g., to run additional code at the same time), click on the blue *R* icon in the top-left corner.

3. type in the Server address:
   `sftp://rstudio-1.cosine.oregonstate.edu`
   (the default *port 22* should be fine),

4. enter your ONID username and password,

5. leave *SSH Private Key* as "*None*", and

6. check the *Add to Keychain* box in order to save your credentials.[3]

You should now be in your `home` folder (e.g., `/home/novakm/`). Take a moment now to bookmark this connection. (In *Cyberduck*, select the *Add Bookmark* option from the drop-down menu.)

In your `home` folder you should see a folder named *R* and, depending on your settings, several "hidden" files and directories.[4] Enter the *R* folder and place whatever files you want inside of it by dragging them in. Be sure to maintain your project's directory substructure so that all relative links between scripts and data work! In fact, you may want (or need) to put much of your whole repository in place (i.e. have your `data`, `code`, and `output` folders and contents all present).

After you've put your scripts in place, jump back to your browser and you should be good to go with a whole lot more computational power at your disposal.

High Performance Clusters

High Performance Clusters (HPCs) consist of many (often hundreds or thousands) of servers that are all networked together. Each server is called a node. You can choose to work on only a single node (which will probably still be faster than your computer), or on several nodes in parallel. The point of using several nodes is that you can use them in parallel (just like we did last class when we used own computer's cores in parallel). The HPC we'll use today is found at `submit.hpc.cosine.oregonstate.edu`.

[3] The option and label for this check-box may be specific to Macs.

[4] The filenames of hidden files and directories all start with a period (e.g., `.bash_profile`). Don't worry if you don't see them as you'll rarely if ever need them. In fact, you typically don't want to delete them, so be careful if you do see them.

`sftp` file transfer

Let's start by using `sftp` to copy files to the HPC.[5] After logging in to OSU's VPN, you can go through the same steps as for the *RStudio Server* except that the server address is:

`sftp://submit.hpc.cosine.oregonstate.edu`.

If the login was successful, you should be in your `home` folder. Again, take a moment to bookmark the connection. This `home` folder will also have a bunch of hidden files (e.g., `.bash`) that you can ignore if they're visible[6], but it may also look empty. Create a folder and copy your scripts into place, again ensuring that your directory structure is correct so that all specified paths will work as in you project's repository.

`ssh` communication and initial `R` setup

Now switch over to your secure shell (`ssh`) client (e.g., *MobaXTerm* for Windows users, or *Terminal* for Mac users[7]). At the $ prompt, type in

`ssh` *`yourONID`*`@submit.hpc.cosine.oregonstate.edu`

whereupon you'll be prompted for your ONID password. You should then see a welcome screen with a new prompt at the bottom of the window:

`[novakm@head ~j]$`

You're likely going to be using *R* a lot, so let's install it into your user resources automatically so it's available each time you login. You can do that by typing the following at the prompt:

`module initadd R`

If you're going to need an *R* package, you'll first need to install it into your home directory. First load the *R* module by typing-in

`module load R`

Then launch *R* by typing-in

`R`

Then type

`install.packages(''`*`package_name`*`'')`

[5] Note that, rather than using `sftp`, you could also use `ssh` and the command `cp ~/Desktop/filename ~/filename` to copy from your desktop to the server.

[6] You can alter *Cyberduck*'s preferences to hide/show the hidden files.

[7] If you want, you can access *Terminal* from within *RStudio*; the tab for it should be in the same window as the console.

The first time you run the `install.packages()` command, you'll be asked if you want to create a personal library. Answer by typing in `y` for yes.
Notice that you're actually in an *R* session. You could, therefore, work away as you might on your computer (but without the benefit of writing scripts).[8]
To get out of *R* and back to the `ssh` prompt, type `q()` for quit.

`ssh` navigation

Now lets find the files we uploaded using the `ssh` view of the cluster. Back at the `ssh`, use the list (`ls`) command:

`[novakm@head ~j]$ ls`

The `ls` command will show you all the sub-directories and files within the directory you're currently in (which at this point in time is your `home` directory). You should see the files (or the folder) you uploaded and, if you installed any *R* packages, a folder named `R_libs`. In order to enter a subdirectory, use the change directory (`cd`) command followed by the name of the directory you'd like to enter:

`[novakm@head ~j]$ cd` *subdirectory*

You can move down into multiple nested dirctories, e.g.,

`[novakm@head ~j]$ cd` *subdirectory/subsubdirectory*

and move out of any number of directories, e.g.,

`[novakm@head ~j]$ cd` *../../otherdirectory*

just as we did when setting relative paths in *R*.

A very useful feature here is that you don't need to type out the whole name of a directory or file. Just type the first letter of its name and then hit your tab key. The name will autofill until it gets to a letter where it can't distinguish between similarly named files. Type the distinguishing letter and tab to continue until you've got the whole name.

Once you're in the directory in which you'd like to be, type `ls` to confirm all the contents are there as needed. If you'd like to take a quick look at the contents of a file, you can use the concatenate (`cat`) command:

`[novakm@head ~j]$ cat` *file.r*

The contents won't be rendered very nicely, but the function is nonetheless useful for confirming the contents of short scripts (such as the `submit.sh` submission script that we'll talk about below).

[8] You can use such command-line interfaces for *Matlab* or *Mathematica* too. In fact, there are many, many other such "modules" available on the HPC. Just type `module avail` to see the whole list.

`ssh` job submission

When performing analyses on an HPC, you (typically) don't do so be entering into the module (e.g., into *R*, the way we did above when installing *R* packages). Rather, you use a submission script to submit your code (your "job") to the cluster. The primary reason for doing that is that the cluster has extensive automated job managment tools which optimize the use of nodes among users and nodes. Therefore, when you submit a job, the cluster (typically) determines which of its nodes it will send the job to. It's similarly so when your job contains code that performs parallelized computations.

The submission script for an *R* job on our HPC will look like this[9]:

```
#!/bin/sh

# Job name (replace R_simple but leave -N)
#$ -N R_simple

#$ -S /bin/sh

# Set working directory on all host to
# directory where the job was started
#$ -cwd

# Send output to job.log (STDOUT + STDERR)
#$ -o job.log
#$ -j y

# Email information (to receive email at process end)
#$ -m e
#$ -M username@oregonstate.edu

#Change which version of R you want to load
module load R/3.3.1

# Command to run (replace test.r but leave Rscript)
Rscript test.r
```

[9]Note that the # symbols are *not* for commenting-out lines of code. They're necessary! (The #$, for example, denotes a command-line argument to be passed to the job scheduler.)

Copy the above into a text file[10], edit it as needed (remembering especially to give the job an informative name at the start and change the script it runs at the end), and save the file with an `.sh` extension (e.g., `submit.sh`).[11] Now upload this file into the same directory as the `test.r` script it calls using your `sftp` client (e.g., *Cyberduck*). Ensure it's in place using your `ssh` client.

Let's assume that our `test.r` analysis script and our `submit.sh` submission script are located in a directory named `mytest`. Use `cd` to change into the directory and `ls` to ensure both your script and your submission script are present.

To submit your job, use the `qsub` command:

`[novakm@head mytest]$ qsub submit.sh`

There are a few commands with which you can view and confirm that your job is running. Each offers different and different amounts of information.

`[novakm@head mytest]$ qstat -u` *username*

You'll see the "state" of your job in the far right column[12]. The most common states for a process to be in are "q" (qeued), "r" (running) and "s" (suspendend), and "e" (error)[13]. To see how much of the cluster's resources your jobs are using, type

`[novakm@head mytest]$ top -u` *username*

For more abbreviated views, use `ps` (for all of your running processes) or `jobs` (for all of your current running jobs).

Job completion and ending a job

If you typed-in your correct email address in the submission script, you'll get an email sent to you when it completes successfully or ends in an error. A log of your job will also be saved to the output file you specified in your submission script (which can be useful for debugging). The log will include output that your script printed to screen (i.e. what would have appeared in

[10] Examples for a *Mathematica* submission and an *R* array submission (distributed over multiple nodes) is provided in the `HPC_examples` folder of today's class folder.

[11] You can give it any name you want, just give it the `.sh` extension.

[12] To have a "live" view rather than a single snapshot, use `watch qstat`. Use command-key period to escape.

[13] To get more information, use `qstat -j`.

your *R* console had you run the script on your computer). Use your `sftp` client to grab all the output your script produced.

Should you decide that something isn't working right for an active job (e.g., a job is taking far longer than expected, eating up too much of the cluster's resources, or you realize you do in fact have a bug in it), you can end it prematurely using the `qdel` command:

1. get its *process ID* (PID) from the left hand column of the `top -u` view;
2. return to the command prompt (using your command key, e.g., Apple key or Windows key, followed by the period key);
3. type `qdel` followed by the process ID.

If `qdel` doesn't work, replace `qdel` with the nuclear option: `kill`.

Finally, to close your `ssh` session, use the `exit` command.

Command	**Action**
Command-key period	Return to command prompt (end current view)
`qstat -u` *username*	View status of submitted jobs
`top -u` *username*	View resource usage by job
`qdel` *process id*	Stop and delete a submitted job
`kill` *process id*	Stop and delete a submitted job (nuclear option)
`exit`	Close your `ssh` connection to the HPC