Introduction to HPC and Parallel Computing

Jing Zeng

Department of Statistics
Florida State University

October 7th, 2019

Linux and Vim

Unix-based OS VS Microsoft Windows

Unix-based operating system:

- FreeBSD: Open-source
- Linux
  - Open-source: Debian, Ubuntu and OpenSUSE.
  - Commercial: Red Hat.
- MacOS: Commercial.

Microsoft Windows: Commercial

Basic command in Linux/MacOS

- Manual command: man
- Print the current path: pwd
- Root path: ~
- Change directory: cd
- List command: ls, ls -l, ls -al
- Make directory: mkdir
- Make new file: touch
- Move a file/ Rename: mv
- Remove file: rm, rm -R (remove the whole directory)
- Retrieve the history: history
- Replay the history: !number (e.g. !2, replay the second command in history)
- Command Completion: [Tab], [Tab][Tab].

Vim

- Vim is a text editor program for Unix.
- Other text editor: Vi.
- Enter Vim editor: `Vim` [filename]
- Basic modes:
  - Normal mode: move, delete, replace
  - Insert mode: everything about editing
  - command mode: search, save, exit
- Switch between each mode:

$$\text{Insert} \underset{\text{i,a,o}}{\overset{\text{Esc}}{\rightleftarrows}} \text{Normal} \underset{\text{Esc}}{\overset{\text{:,\textbackslash,?}}{\rightleftarrows}} \text{Command}$$

- Normal
  - `h, j, k, l`: $\leftarrow, \downarrow, \uparrow, \rightarrow$.
  - `[Ctrl]+f`, `[Ctrl]+b`: move one page forward, move one page backward.
  - `G`, `gg`: move to the last line, move the first line.
  - `$`, `0`: move to the end of the current line, move to the start of the current line.
  - `x`, `X`: delete forward, delete backward
  - `dd`: delete a whole line
  - `r`: replace a character.
- Insert
  - `i`: enter insert mode from the current cursor.
  - `a`: enter insert mode from the character after the cursor.
  - `o`: enter insert mode from the next line.
- Command
  - `?+[word]`, `/+[word]`: search for the `[word]` forward, search for the `[word]` backward.
  - `:q`, `:w`, `:wq`, `:q!`: quit, save, save and quit, quit without saving.

High-Performance Computing

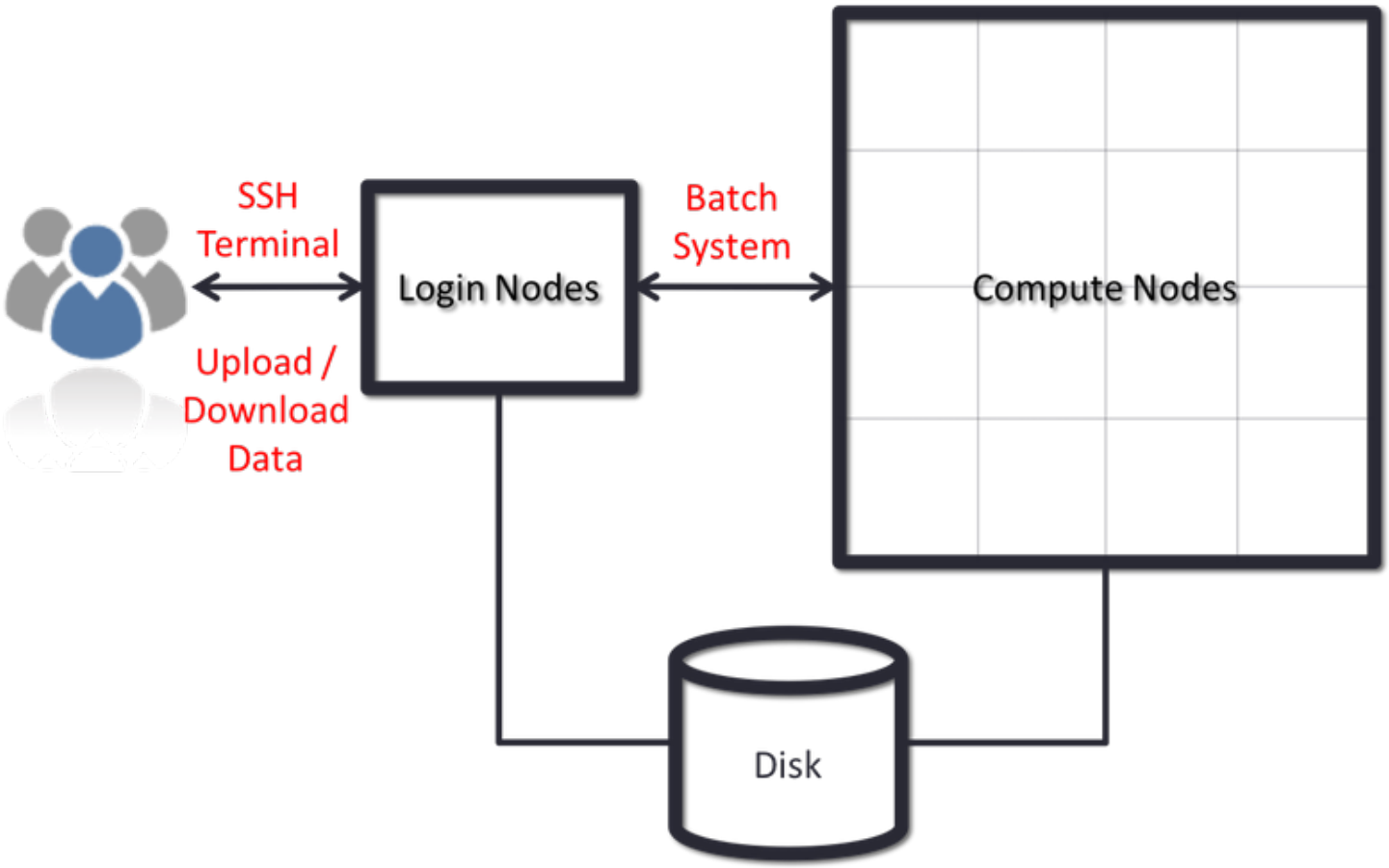
Why use HPC?

- Program is time-consuming or memory-intensive.
- A lot of programs need to run.
- HPC is efficient, reliable and fast.
- A skill one has to control if the future career are professor, data scientist or engineer.
- Just for fun.

Basic definitions

- Partition:
  - Queue.
  - Scheduler: allocate the jobs according to node, CPU, or core requirements
- Node: a physical computer.
  - Login node
  - Computing node
- CPU: Computing node normally consists of 2 CPUs.
- Core: each CPU normally consists of 2 to 16 cores (AMD, Intel).
- Logical core: multiple processes in each core, perform like actual physical cores in practice.

Nodes and scheduler

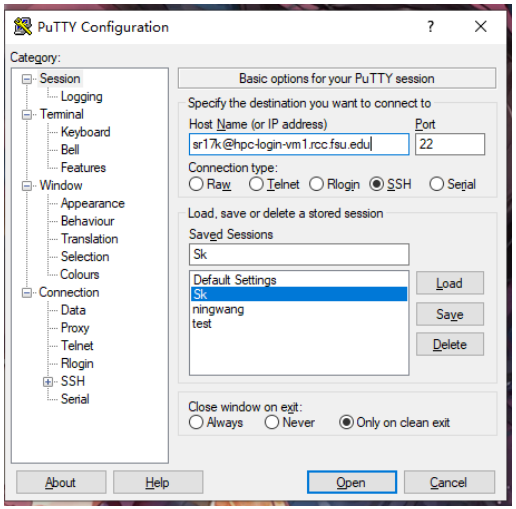


Workflow

- Input datasets → HPC system (login nodes)
- Write a job submission script (login nodes)
- Submit job submission script → partition (login nodes)
- Program is running(computing nodes)
- Output results (computing nodes)

Login and logout

- Login
  - Macbook: `ssh [fsuid]@hpc-login.rcc.fsu.edu`
  - Windows: Putty + ssh




```
(base) cengjingdeMacBook-Air:~ JingZeng$ ssh jz17c@hpc-login.rc
c.fsu.edu
jz17c@hpc-login.rcc.fsu.edu's password:
Last login: Sat Sep 28 16:52:18 2019 from fsu-38d95.vpn.fsu.edu

Welcome to the RCC
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
RCC/HPC Documentation can be found here:
https://rcc.fsu.edu/docs
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


** Disk usage (GPFS) quota report: 5.438G used of 135G availabl
e
For a disk quota report, run: gpfs_quota


-bash-4.2$
```

- Logout: `exit`

Login and logout with VPN

If you are using non-FSU Network, e.g., when you are working at home, you need an off-campus VPN to connect to HPC.

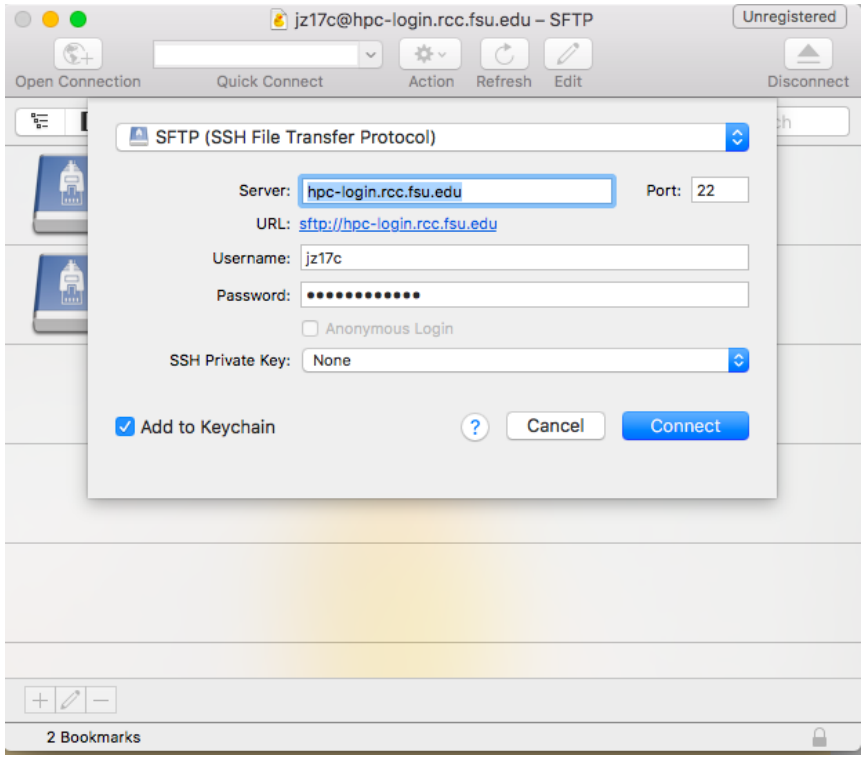
For details:
`https://rcc.fsu.edu/doc/off-campus-vpn-access`

Transfer of files

- Macbook: Use SFTP on the Terminal:

Connect `sftp [fsuid]@export.rcc.fsu.edu`
Send `put [local file] [server]`
Receive `get [server file] [local]`

- Windows and Macbook: Use Cyberduck



How to write a bash script/job script

- Example: Rrun.sh

```
#!/bin/bash

#SBATCH --job-name="Example"
#SBATCH -N 1
#SBATCH -n 8
#SBATCH -p statistics_q
#SBATCH --mail-type="END"
#SBATCH --mail-type="FAIL"
#SBATCH -o %A.out
#SBATCH -e %A.err

module load R
Rscript Code.R
```

How to write a bash script/job script

- `#!/bin/bash`: the shell interpreter
- `#SBATCH`: Slurm options
  - `-N`: the number of nodes (computers)
  - `-n`: the number of cores
  - `-p`: the partition name
  - `--job-name/-J`: the name of the job
  - `mail-type`: when the mail will be sent
- `module load R`: R is loaded into the environment
- `Rscript` + [Rcode file]: specify the code which is going to run

Submit the job

Files we need:

- bash script/job script: `Rrun.sh`
- R code: `Code.R`

sbatch Submit the job: `sbatch` [bash script]

```
-bash-4.2$ sbatch Rrun.sh
Submitted batch job 617653
```

Note

Slurm options in file is more like the default parameters, we can overwrite it in command.

```
sbatch -p genacc_q --job-name Stat -N 2 Rrun.sh
```

We changed the partition to `genacc_q`, rename the job name and reset the number of nodes.

Track the status

Useful slurm commands:

squeue Check the queue/partition:
squeue -u [user id], squeue -p [partition].

```
-bash-4.2$ squeue -u jz17c
JOBID   PARTITION   NAME    USER  ST  TIME  NODES
617653 statistic  Example  jz17c PD 0:00  1 (Priority)
```

scancel Cancel the jobs: scancel [job id].

```
-bash-4.2$ scancel 617653
```

Track the status

sinfo Check the information of partition: sinfo -p [partition].

```
-bash-4.2$ sinfo -p statistics_q
PARTITION    AVAIL  TIMELIMIT  NODES  STATE NODELIST
statistics_q    up   infinite      1    mix hpc-d33-7-2
statistics_q    up   infinite      4  alloc hpc-d33-7-[3-4],hpc-d33-8-[1-2]
```

sacct Displays information on jobs, status, and exitcodes by default: sacct -u [user id], sacct -p [partition].

```
-bash-4.2$ sacct -u jz17c
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
617155          Example statistic+ statistic+          8 CANCELLED+      0:0
617653          Example statistic+ statistic+          8 CANCELLED+      0:0
```

Job arrays

Used to submit a large number of **independent** jobs.

Example

We have 100 R codes, Code1.R, Code2.R, . . ., Code100.R, where in each code we have different simulation data or we want to try different parameters.

```
#!/bin/bash

#SBATCH --job-name="Example"
#SBATCH -N 1
#SBATCH -n 8
#SBATCH -p statistics_q
#SBATCH --mail-type="END"
#SBATCH --mail-type="FAIL"

#SBATCH -o %A_%a.out
#SBATCH -e %A_%a.err
#SBATCH --array=1-5

module load R
Rscript Code${SLURM_ARRAY_TASK_ID}.R
```

Job arrays

- `--array`: specify the array id
- `%A`: master job id
- `%a`: array task id
- `${SLURM_ARRAY_TASK_ID}`: the slurm environment variable, corresponding to `--array`.

```
-bash-4.2$ sbatch -p genacc_q Rrun_array.sh
Submitted batch job 618404
```

```
-bash-4.2$ squeue -u jz17c
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
         618404_5  genacc_q  Example    jz17c  R       0:00      1 hpc-d36-3-2
         618404_1  genacc_q  Example    jz17c  R       0:01      1 hpc-d30-10-2
         618404_2  genacc_q  Example    jz17c  R       0:01      1 hpc-d30-10-2
         618404_3  genacc_q  Example    jz17c  R       0:01      1 hpc-m32-10-1
         618404_4  genacc_q  Example    jz17c  R       0:01      1 hpc-d36-3-1
```

Resources allocation

- Preparation for parallel computing.
- Consider: how long your job needs to run → how many compute cores and nodes are needed (more things to consider, e.g., GPU, memory)
- Important slurm options:
  - `-n`: The number of cores.
  - `-N`: The number of nodes (computers).
  - `--cores-per-socket`: the minimum number of cores the processor (CPU) is supposed to have.

Example

#SBATCH –cores-per-socket=8
#SBATCH -n 8
Request 1 CPU with 8 cores.

Strategy

- If we asked for too many cores, it is sometimes not easy for the HPC scheduler to find such available resources. In this case, you could reduce the number of cores.
- Don not ask for more than one node, it will not speed your code typically.
- If you ask for a CPU with at least 8 cores, it is not easy for the HPC scheduler to find such a CPU with all cores available. In this case, you would better remove this request.

Parallel Computing

Rewrite the simulation with lapply

For the **independent** iterations (e.g., replicates in simulation), try to avoid `for` loop and make the most of `apply` family (`apply`, `sapply`, `lapply`, `tapply`, `vapply`, `replicate`).

- No `for` loop
  - Slow, inefficient
  - Copy the objects in each loop, memory is not released after each loop.
  - Any `for` loop can be rewritten with `sapply` or `lapply`.
- Rewrite the code with `sapply` or `lapply`
  - If the output in each replicate is a vector, use `sapply` or `lapply`
  - If the output is not a vector or consists of several objects, use `lapply`
  - In parallel computing, use `parallel::mclapply`, which is almost the same as `lapply`.

Minimal workable example (MWE) 1

- MWE for `sapply` function

```
# mwe for sapply
result1 <- sapply(1:10, function(i){
   return(rnorm(100))
})
```

```
> a <- read.table('mwe1')
> dim(a)
[1] 100  10
```

- MWE for `lapply` function

```
# mwe for lapply
result2 <- lapply(1:10, function(i){
   return(rnorm(100))
})
save(result2, file='mwe2')
```

```
> load('mwe2')
> class(result2)
[1] "list"
> length(result2)
[1] 10
```

Parallel computing

- Use `parallel::mclapply` function, set the number of cores, `mc.cores`.
- Detect the available cores: `parallel:detectCores`

```
> detectCores()
[1] 16
> detectCores(logical=FALSE)
[1] 16
```

- Reproducibility:

```
RNGkind("L'Ecuyer-CMRG")
set.seed(1)
```

Minimal workable example (MWE) 2

Sleep 30 seconds in each replicate.

```
# mwe for mclapply
result3 <- mclapply(seq_len(1000), function(i){
    cat('Time', i, '\n')
    Sys.sleep(30)
    rnorm(100)
}, mc.cores=8)
```

Output file:

```
Time 2
Time Time4
3
Time 1
Time 7
Time Time8
5
Time 6
```

Other functions

- Other libraries or functions to do parallel computing in R: `foreach`, `doParallel`, `parLapply`, etc.
- Useful links:
  - `https://nceas.github.io/oss-lessons/parallel-computing-in-r/parallel-computing-in-r.html`
  - `http://dept.stat.lsa.umich.edu/~jerrick/courses/stat701/notes/parallel.html`

Thank you!