R on Biowulf

Case studies with Q & A

Qi Yu, Wolfgang Resch

Biowulf, NIH

2022-11, updated on 2024-08-21

R on Biowulf

This is not an introductory R class. This is for users who are familiar with R on their desktop or other servers.

This class aims to help you to:

- migrate smoothly to use R on Biowulf
- submit batch R jobs to the cluster
- run your R code efficiently

through some case studies.

Case 1

Fry has a R job that needs larger memory than is available on his laptop. He plans to transfer his code and data to Biowulf to run his job on the cluster.

What does he need to finish his job on biowulf?

- Biowulf storage (/data - individual or shared directory)
- Temporary local scratch lscratch
- Select an R version
- R package library management
- R job submission

Storage

Fry should make sure his Biowulf account has enough storage space in /data for inputs and results.

```
$ checkquota
Mount                   Used      Quota  Percent    Files    Limit  Pe
/data:              256.0 KB   100.0 GB    0.00%        0 31457280
/home:               24.0 KB    16.0 GB    0.00%        7      n/a
```

/home (16GB, fixed size)

default:

/data (100GB, expandable -- request quota increase by filling out the online form)

There are several ways to transfer data

Using lscratch for R

Fry should request temporary local scratch for R interactive sessions or batch jobs on HPC.

He allocates 20G lscratch for interactive session:

```
$ sinteractive --gres=lscratch:20
```

He checks if `lscratch` is used in R:

```
> Sys.getenv('TMPDIR')
"/lscratch/63315403" # Good. The number represents the job id.
"/tmp" # Without lscratch
```

1. R will automatically use lscratch for temporary files if it has been allocated on Biowulf.
2. lscratch > 2GB is recommended.
3. lscratch directories are job-specific and cleaned up when a job is done.

R versions

R is installed as a module on Biowulf, and updated frequently.
`module load R` will load the default version (currently R/4.4).

```
$ module spider "R"
R/4.0.0, R/4.0.3, R/4.0.5, R/4.1.0, R/4.1.3, R/4.2.0...
```

Each verion includes many common R packages.

If Fry needs different version of packages for multiple projects, it's better to use packrat: a dependency management system or renv.

He checks the R version and loaded packages:

```
> sessionInfo()
```

1. sessionInfo() is very helpful for troubleshooting.

Managing R packages

A large number of R packages are installed with R on Biowulf. Users can also install packages themselves. By default, Fry's own R packages will be installed to `/data/$USER/R/rhel8/%v`. He has to create this directory himself in bash. He will need to know the R version. For R/4.4*, the code would be:

```
$ mkdir -p /data/$USER/R/rhel8/%v
```

R will search libraries in this order:

```
> .libPaths()
[1] "/data/Fry_username/R/rhel8/4.4"
[2] "/usr/local/apps/R/4.4/site-library_4.4.1"
[3] "/usr/local/apps/R/4.4/4.4.1/lib64/R/library"
```

1. All the local R/4.2* will share the same library location.
2. The library for each version of R only need to be created once.

Reinstalling R packages

To replicate his local environment, Fry first check the version of Bioconductor to make sure it is the same as cluster.

```
>BiocManager::version()
[1] '3.19'
```

Then Fry gets the list of installed packages on his local computer:

```
> packages <- installed.packages()[,"Package"]
> save(packages, file="Rpackages.rda")
```

1. Two Bioconductor versions are associated with one specific R versions.
2. Attempting to install a version of Bioconductor that is not supported by the version of R in use leads to an error.
3. Using the most recent version of Bioconductor may require installing a new version of R.

Reinstalling R packages

He transfers Rpackages.rda to Biowulf.

Fry then loads the RData file and installs:

```
> load("Rpackages.rda")
> toInstall <- setdiff(packages, installed.packages()[,"Package"])
> BiocManager::install(toInstall)
```

This will download and install the R libraries from Bioconductor and CRAN.

When Fry runs `BiocManager`, it will always check updates for all packages (including the system packages). Fry should choose "n" since he doesn't have the ability to update the system packages.

```
Update all/some/none? [a/s/n]: n
```

1. Always use BiocManager to install Bioconductor packages

How about other packages?

Install package from GitHub:

```
> devtools::install_github("trinker/pacman")
```

Biowulf has 1797 R libraries installed. If the package is available on CRAN or Bioconductor, then Fry can email staff@hpc.nih.gov to install it.

`R CMD INSTALL` can be used for installing add-on packages.

Testing a library without modifying the local library

pacman is a R package management tool, it conveniently wraps library and package related functions and names them in an intuitive and consistent fashion.

```
> library(pacman)
> p_temp(rapport)
```

If everything looks good, then he can install with:

```
> p_install(rapport)
```

Other useful pacman functions: `p_loaded`, `p_unload`, `p_install_gh`, `p_path`, `p_iscran`.

Fry then submits R jobs to the batch scheduler

`Rscript` (non-interactive variant of R command)

Example: 'rjob.sh'

```
#!/bin/bash
#SBATCH --job-name=R-job     # create a short name for the job
#SBATCH --cpus-per-task=1    # cpu-cores per task (>1 if multi-threade
#SBATCH --mem=4G             # memory
#SBATCH --time=04:00:00      # total run time limit (HH:MM:SS)
#SBATCH --gres=lscratch:20    # using local disk
module load R
cd /data/Fry_username/
# Use Rscript
Rscript Rcode.r > Rcode.out
```

He submits the job with sbatch:

```
$ sbatch rjob.sh
```

Questions and Brake

05:00

Case 2

Fry left NIH. His colleague Leela inherits his code, but can't run this code successfully. The error is from `Seurat` package, so she writes to staff@hpc.nih.gov for help.

What information can Leela provide for troubleshooting?

- Error details and jobID (if it's a batch job)
- R version (`sessionInfo()`)
- Library versions
- Is it a local or system package?

Leela checks the version and location of loaded libraries

```
> packageVersion('Seurat')
[1] ‘5.1.0’
> library(Seurat)
> path.package('Seurat')
[1] "/data/Leela_username/R/rhel8/4.4/Seurat"
```

Or use pacman:

```
> library(pacman)
> p_version(Seurat)
[1] ‘5.1.0’
> p_path(Seurat)
[1] "/data/Leela_username/R/rhel8/4.4/Seurat"
```

Leela found that her `Seurat` library is loaded and version `5.1.0`.

What should Leela test first?

She switches to the system version instead of her local version:

```
> p_unload(Seurat)
> library(Seurat, lib.loc="/usr/local/apps/R/4.4/site-library_4.4.1/'
```

1) If the error persists, she writes to staff@hpc.nih.gov with the error, a replicable example, and with the output of `sessionInfo()`.

2) If the error is gone, it's time to remove her local packages since that would has the priority to get use, or assign lib.loc in the code to use system version.

Removing Leela's local installed packages

```
> remove.packages("test_package")
```

Or with pacman:

```
> library(pacman)
> p_delete(test_package)
```

Or she deletes the directory with that package:

```
$ rm -r /data/$USER/R/rhel8/<ver>/test_package
```

Testing R interactively?

Leela plans to plot some figures with the data on Biowulf, she wonders: is there other way (besides command line) to run R interactively?

- RStudio (https://hpc.nih.gov/apps/RStudio.html)
- RStudio Server with tunneling (https://hpc.nih.gov/apps/rstudio-server.html)
- RStudio Sever on HPC OnDemand (https://hpcondemand.nih.gov)
- jupyter (https://hpc.nih.gov/apps/jupyter.html)

05:00

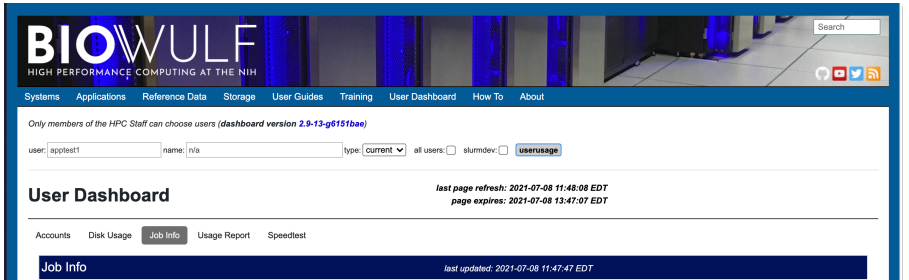
Case 3: Will more CPUs and/or more memory = less running time?

Amy has a deadline coming soon. She wonders if allocating more CPUs and memory will speed up her job?

It depends. Base R is mostly single-threaded. Regardless how many cores are allocated, many R scripts can only use one CPU unless the code was explicitly written to use more than one CPU. As for memory, most of the time the additional resources are not used.

```
$ dashboard_cli jobs --compact
jobid     st  submit_time  partition  n  c  m     timelimit  gres
=============================================================================
16340213  CD  02T09:19:33  norm   1  8  10GB    8:00:00  lscratch:5
```

Alternatively, check CPUs and memory usage on graphic dashboard.



When to increase memory?

- Amy's job failed with an error indicating that a process was killed:

```
/var/spool/slurm/slurmd/job10000101/slurm_script: line 82: 39920 Kil
```

- Or, if Amy was running Rstudio, her R Session was killed.
- Or the dashboard shows:

jobid	jobname	state	statetime	nodelist	eval
18019104	sinteractive	COMPLETED	2021-06-28 19:58:04 EDT	cn0890	⚠

- Or the dashboard_cli shows `MEM_OVER`:

```
$ dashboard_cli jobs -u apptest1 --since 6/20 --compact
jobid     st  submit_time  partition  n  c   m      timelimit   gres
=============================================================
18019104  CD  28T12:45:27  interact+  1  2  2GB     8:00:00     -
```

How does Amy know if parallel computing is worth to try?

- Her jobs run for a long time (e.g. 9 days) - Yes
- Her R code is written in Rcpp - No
- One part of the code consumes most of the runtime - Yes

How does she identify the "bottleneck" in the program?

- system.time()
- `Rprof()` and `summaryRprof()`
- Rprof() runs the profiler for performance of analysis of R code.
- summaryRprof() summarizes the output of Rprof and gives percent of time spent in each function

How does Amy know if her code can run in parallel?

- levels of parallelization: multiprocessing vs multithreads
- Can her job be split to multiple independent processes?
- Does a function in her code support multiple threads?
- Is there 'lapply/sapply' function?
- Is there 'for' loop?

Running multiple (pleasingly parallel) single-thread R jobs concurrently

Amy can submit swarm jobs if each R process is independent.

Generate R.swarm:

```
Rscript countA.R
Rscript countB.R
Rscript countC.R
```

Submit to Biowulf:

```
$ swarm -f R.swarm --gres=lscratch:20 --module R/4.4
```

Common pitfall: Failed R jobs with "all connections are in use" error

Amy's swarm jobs were submitted to the cluster, but *some* of them failed.

She wonders why?

- Some of the Biowulf nodes have 128 CPUs (phase6 nodes) or 144 CPUs (*largemem* nodes), check with `freen`
- some R packages will detect all cores on a node even if they are not allocated (e.g. `parallel::detectCores()`)
- If not explicitly specifying the number of CPUs R may try to use *all* CPUs but R can't communicate with more than 128 parallel worker processes
- Solution: use `parallelly::availableCores()` to detect allocated CPUs and use fewer CPUs

Can Amy's code use multiple threads?

R on biowulf uses the Intel MKL math libraries which can use multiple threads for some functions. Amy can find out if her script could speed up by allocating more CPUs and comparing the runtime between single-threaded and multi-threaded:

```
$ sbatch --cpus-per-task=4
```

And she uses the `OMP_NUM_THREADS` variable in her script:

```
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
Rscript script.R
```

She checks the runtime and CPU utilization on the dashboard and found out she was using 32 CPUs (overloading) instead of 4 CPUs.

Overloading could dramatically slow down your job

So she change the `OMP_NUM_THREADS` to 1 (which is also the default setting of R module on Biowulf):

```
export OMP_NUM_THREADS=1
Rscript script.R
```

She checks the runtime and CPU utilization on the dashboard and found out she was using only 1 CPU instead of 4 CPUs.

Can Amy modify the code to use more CPUs?

Amy replaces 'lapply' with 'mclapply' and setup 'mc.cores' to use multiple worker processes to parallelize work.

Example with 12 CPUs:

```
> library(parallel)
> ncpus <- parallelly::availableCores()
> options(mc.cores = ncpus) # set a global option for parallel packag
# Then run mclapply()
> mclapply(X, FUN, ..., mc.cores = ncpus)
```

1.Similarly, 'sapply' could be replaced by 'mcsapply'.

Performance comparison between lapply() and mclapply()

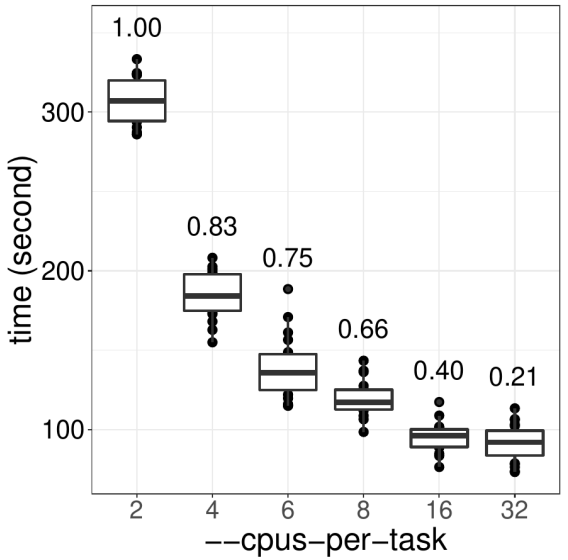
```
> N <- 10^6
> system.time(x<-lapply(1:N, function(i) {rnorm(300)}))
##   user  system elapsed
## 36.588   1.375  38.053
> system.time(x<-mclapply(1:N, function(i) {rnorm(300)},mc.cores = nc
##   user  system elapsed
## 11.587  14.547  13.684
```

12 workers, 3x speedup, efficiency = 3/12 = 25%.

Parallel jobs should aim for an efficiency of 70-80%.

More CPUs do not always run code efficiently

- e.g. running 20 loops with 40 CPUs.
- Benchmark tests are highly recommended for parallel jobs.
- The performance of mclappy() with 2-32 CPUs and compared their efficiency:



Run this R code no more than 6 CPUs for efficiency

1.mclapply/mcsapply can use multiple cores on one node, but not on multiple nodes.

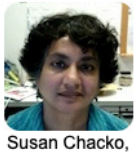
Questions and Brake

05:00

Contact the HPC team with questions

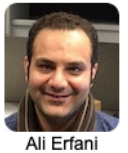
- Always check the up-to-date R doc.
- Email: staff@hpc.nih.gov
- Join our monthly walk-in virtual consultation.


Steve Bailey


Susan Chacko, Ph.D.


Gennady Denisov, Ph.D.

Picture unavailable
Afif Elghraoui


Ali Erfani


Andrew Fant, Ph.D.


Jonathan Goodson, Ph.D.


David Hoover, Ph.D.


Sam Ijeomah


Patsy Jones

Picture unavailable
Charles Lehr


Jean Mao, Ph.D.


Tim Miller


Nitish Narula, M.S.


Charlene Osborn


David O'Brien


Mark Patkus


Wolfgang Resch, Ph.D.


Antonio Ulloa, Ph.D.


Qi Yu, Ph.D.

Working through a case study 4:

What steps need to be modified to Bender's code with multiple CPUs:

```
library(doParallel)
library(doMC)
registerDoMC(cores=parallel::detectCores())
max.eig <- function(N) {
     d <- matrix(rnorm(N*N), nrow = N)
     E <- eigen(d)$values
     abs(E)[[1]]
 }
for (n in 400:520) max.eig(n)
```

batch job:

```
#/bin/bash
module load R
R --vanilla < test.R >test.out
```

submit with:

```
$ sbatch test.sh --ntask=6 --mem=4
```

01:30

One solution for case 4:

Code:

```
library(doParallel)
library(doMC)
registerDoMC(cores=parallelly::availableCores())
max.eig <- function(N) {
     d <- matrix(rnorm(N*N), nrow = N)
     E <- eigen(d)$values
     abs(E)[[1]]
 }
foreach(n = 400:520) %dopar% max.eig(n)
```

batch job:

```
#!/bin/bash
module load R/4.2
cd /your_directory/ # or assigning full path to test.R
Rscript test.R >test.out
```

submit with:

```
$ sbatch --gres=lscratch:20 --mem=4g --cpus-per-task=6 test.sh
```

Working through a case study 5:

- Rscript run_sim_1.R sim_1.out
- Rscript run_sim_2.R sim_2.out
- ...

01:30

How would Bender modify the code to be able to run on the cluster:

(hint: check our R doc):

- Rscript run_sim.R 1 sim_1.out
- Rscript run_sim.R 2 sim_2.out
- ...

Q & A summary

Q: Should I always grab a lot of memory and CPUs for my sinteractive sessions?

A: Only use the resources required for sinteractive sessions. Sinteractive sessions, like batch jobs are exclusive to you and idle resources from an sinteractive session will not be available to other jobs/users.

Q: How does lscratch work?

A: local scratch is a job-specific directory on the local SSD of a compute node. It contains the jobid in its path (`/lscratch/$SLURM_JOB_ID`) and is deleted at the end of a job / sinteractive session. Any data in your jobs lscratch that you want to save has to be explicitly copied back to your data directory. lscratch is used for temporary files by R automatically if it has been allocated. See our lscratch docs

Q: How do we know how much lscratch to allocate?

A: If you aren't using lscratch explicitly in your code with R then usually 5GB is plenty. However some R packages create a lot of temp files so you may need more.

Q: what is the limit for per job lscratch size?

A: Currently the largest lscratch volume you can use is 3.2TB but this may change in the future. Use the `freen` command to check the up-to-date size of lscratch as well as memory and CPUs for all our compute nodes.

Q: Is there a way to profile your code to estimate temp storage?

A: We currently don't keep automatic track of lscratch usage for jobs but there is a script called `lscratch_mon` that you can run in the background if you want to profile a job

Q: Can you explain what the data directory is please? Is it a personal directory? What is the benefit of /home vs /data

A: See our storage docs. /home is for notes, code, config files, state files, programs you install, caches, ... It has a fixed quota of 16GB. /data is for your data and analysis outputs. Every user has a personal data directory at `/data/$USER`. Some users also have access to one or more group shared directories (`/data/GroupName`).

Q: How can I use R as interactive mode with R studio?

A: See our Rstudio docs, Rstudio Server docs and HPC OnDemand.

Q: how about future, rslurm, Rmpi, clustermq for parallelism

A: All are available on biowulf but we don't yet have much detailed documentation except for Rmpi. If you use Rmpi please carefully evaluate parallel efficiency of your code. We will work to improve our documentation.

Q: If I use Rstudio GUI in biowoulf, and I need to use some packages, so every time should I have to install everytime ?

A: No. Once you install a package that is not available with our R install already you don't have to reinstall. You may have to updated at some point.

Q: as a user I can’t update R packages and only use the latest version installed by the system administrator? If I need a newer package version I have to reach out to IT?

A: Your personal R library takes precedence over the centrally installed packages so if you install a newer version of a package it will be loaded preferentially.

Q: If you want a graphical interface, you can also use Jupyter notebooks on Biowulf

A: True. R kernels for our R installs are available in our jupyter install.

Q: Is there any Rstudio server available at Biowulf?

A: Yes, Rstudio Server docs and HPC OnDemand.

Q: I thought the max run time on a node was 24 hours?

A: Different partitions have different max. walltimes. sinteractive is 36h. norm is 10 days. see the output of `batchlim` for up-to-date information.

Q: Can you increase the mem limit while within an interactive session?

A: No. On biowulf you can't increase/decrease memory or CPU for a running job/sinteractive session.

Q: How do you specify multithreads vs multiprocess?

A: That's a difficult topic. The most common form of parallelism in R is multiprocessing. This is usually explicitly done by you or a package you are using. There are are some parts of base R and the underlying math libraries that can multithread which is mostly implicit parallelism. You can check if your code can take advantage of that. You can allocate for example 4 CPUs and then run your script with different settings of the `$OMP_NUM_THREADS` or `$MKL_NUM_THREADS` environment variable. If you see a significant speed up and the dashboard data shows that it used multiple CPUs then it's worth using more than one CPU.

It is important to always test parallel efficiency and monitor actual usage of CPUs and memory with the dashboard (https://hpc.nih.gov/dashboard) or using the `dashboard_cli` command. For running jobs there is also `jobload`.

Q: Can we use a for loop instead of duplicate lines in swarm file?

A: Swarm files are line oriented. Theoretically you can squeeze a for loop into a single line in a swarm file but I'm not sure that's the right way to go

Q: If I want to run a simulation, nsim=100 but each simulation takes a while (~30minutes). Would the best approach be to break up the script so each .R script runs a batch of 10 or something and then run those R scripts using R.swarm or is there a better way to do this?

A: That would certainly be one way to do it. Since each simulation runs for 30 minutes you could also break that up into a swarm of 100 single simulations. That is ok in this case b/c we generally want jobs to run for > 15 m. If your simulations were very short running we would ask that each job do multiple simulations as you propose.