Practical guide to running R on HPC

Elzbieta Lauzikaite

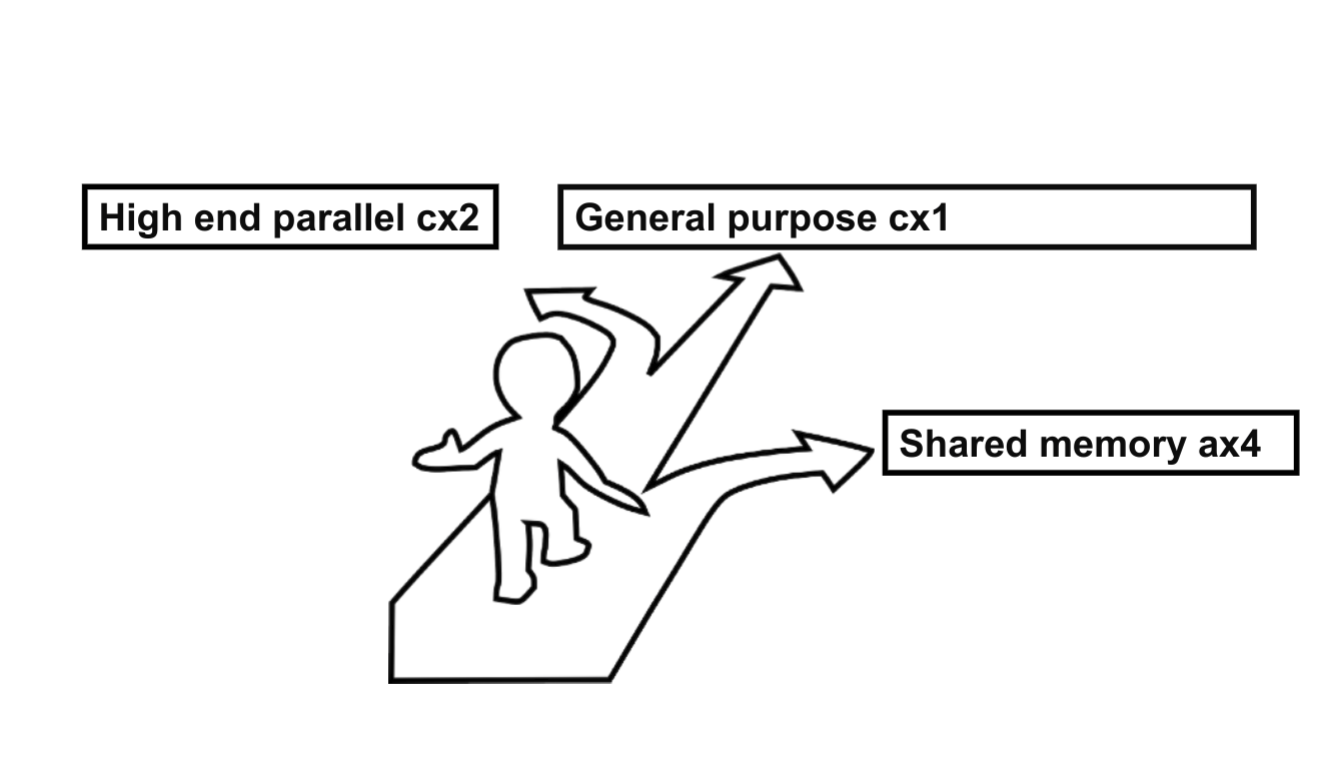
5/3/2019

Table of Contents

# Imperial HPC systems

Imperial has three cluster systems, which are designed for different kind of computations. A computer cluster is system of loosely or tightly connected independent computers (**nodes**). These nodes are managed by a **batch system** (i.e. non-interactive processing mode), which monitors available resources (**NCPUS** and **memory**) and allocates users’ tasks (**jobs**) to available nodes. If none are available, jobs are put on hold in a **queue**, together with the jobs of other users, until sufficient resources are free to use.

User should select a cluster system appropriate for the planned work.



## CX1

## AX4

# Set up

## User account

To access any of the HPC systems, an account must first be created. A request must be made by your group leader through . This will give you access to CX1 free-of-charge, for usage of AX4/CX2, contact HPC/RCS support team.

## Log in

All HPC systems can be accessed through the ssh command via terminal (Linux/Mac) when connected to Imperial VPN:

ssh username@login.cx1.hpc.ic.ac.uk  
ssh username@login.ax4.hpc.ic.ac.uk  
ssh username@login.cx2.hpc.ic.ac.uk

Windows users can use software .

After login, every user is connected to the **login node**, which can accommodate a large number of users, but does not support heavy calculations. Therefore running applications directly on this node can crash the whole system, or at least terminate your connection and your work. Use this node to perform light work, such as data management, script preparation and job submission.

## File management

Decision on data storage should be made as early as possible, since its set up can significantly prolong the start of your work.

### Local HPC storage

For smaller data sets to be processed on CX1, the general advise would be to use local HPC storage. Aliquoted free-of-charge space is usually sufficient and can be extended further through the self-service portal. However, note that these .

Files can be copied over to the HPC storage by:

1. Secure copy command

scp /files.tgz username@login.cx1.hpc.ic.ac.uk:/home/username/file\_location

1. FileZilla on Mac and Windows. FileZilla is also handy when managing your scripts and job output on the HPC.

### Remote storage

There is a number of options for remote data storage, though the default BOX and H:drive are not appropriate for sensitive data either.

# Preparing working environment

A large number of applications are already available on the systems. They are centrally installed and accessible by every user through *module* command. Check whether your application is available through *module avail* command on the login node.

For some applications and libraries, additional modules need to be loaded. Read more about modules on .

To start R for the use of XCMS, the following modules need to be loaded during your session:

module load intel-suite libxml2 hdf5/1.8.14-serial netcdf/4.4.1 R/3.4.0  
module load boost

To load a centrally available R library, define the path to their location:

.libPaths("/apps/R/3.4.0/lib64/R/library", .libPaths())  
library(xcms)

While most popular R packages are already available on CX1, AX4 has fewer of them. You may need to install the missing ones yourself locally on your HOME directory while running R interactively on the login node. Make sure that **all supporting modules are loaded to your enviroment** before you start R. If a package is installed successfully, then you would load it by defining the path to your HOME. Some packages are hard to install locally due to the requirement of additional Linux libraries. In such case, submit a request to the .

.libPaths("/home/username/R/x86\_64-pc-linux-gnu-library/3.4", .libPaths())  
library(xcms)

# Submitting jobs

Management of users’ jobs is done by the PSBPro(Portable Batch System Professional) queue system.**PBS scheduler** allocates every job to nodes/cores according to the amount of resources requested by the user. The scheduler starts the job when sufficient resources are available, runs it and returns the output to the user.

Jobs are submitted via a **PBS script** - a bash script. Such script must include the following information: time of processing (*walltime*), required memory (*mem*) and number of nodes and cores (*select*, *ncpus*). If the requested resources are exceeded during the job run, the job is terminated. Only walltime now can be extended for running jobs via the self-service portal.

A PBS script *run.pbs*:

#!/bin/sh  
#PBS -N job\_name  
#PBS -l walltime=01:00:00  
#PBS -l select=1:ncpus=1:mem=50gb

Submitting *run.pbs* via terminal:

qsub run.pbs

If job was submitted succesfully, a job ID will be returned. Monitor submitted jobs:

qstat   
 qstat -u username # all user's jobs  
 qstat â€“f job\_id # returns more details on a single job

Once job is completed, it will be gone from the *qstat* list. You can also delete uncompleted jobs at any stage:

qdel job\_id

## Resource estimation

Before performing computations are the full-scale, it is advised to monitor resource usage on a small-scale. E.g. apply your code on just a single datafile at once, or run the code in a serial mode to get the estimate for a single **worker** before initiating a job with multiple parallel-workers. To observe memory usage, one can define a parameter in the PBS script to send an email if the code is aborted by the batch system(a), when execution begins(b) or ends(e):

#!/bin/sh  
#PBS -m abe

For parallel code on CX1, a more detailed memory usage can be performed using module *memusage*. The application will save the output in the current directory. The generated output files can be plotted using module *gnuplot*, more details can be found on :

#!/bin/sh  
#PBS -l walltime=01:00:00   
#PBS -l select=1:ncpus=1:mem=1gb  
cd $PBS\_O\_WORKDIR  
/apps/memusage/memusage R

## Submitting R jobs

The best approch for running R scripts in batch mode is **Rscript**. Rscript logs all output and allows to submit command line arguments, both of which are essential when building an automated PBS-based pipeline. Variables, such as datafiles or function parameters, can be supplied as arguments, while re-using the same R code. This helps to perform optimisation tasks or carry out the same workflow with different datafiles.

A PBS script, which starts a serial R job with one argument, while capturing errors and details of the process in an output Rout file:

#!/bin/sh  
#PBS -N job\_name  
#PBS -l walltime=01:00:00  
#PBS -l select=1:ncpus=1:mem=50gb  
module load intel-suite libxml2 hdf5/1.8.14-serial netcdf/4.4.1 R/3.4.0  
module load boost  
Rscript --no-save --no-restore --verbose your\_code.R "args" > "/outfile.Rout" 2>&1

The corresponding *your\_code.R* script:

args = commandArgs(trailingOnly=TRUE)  
argument <- paste0(args[1])

# Running R for XCMS on CX1

The most computationally demanding step of LC-MS data pre-processing with XCMS is initial peak-picking, which requires to load raw datafiles into R memory. Even though peak-picking algorithm is normally applied to all files at the same time, it is an “embarrassingly parallel”" task, since each datafile is processed separately inside the code and then the final single R object is obtained.

Large LC-MS datasets can be run on CX1 through the use of **arrays jobs**. An array job comprises of multiple independently running R sessions, which for the sake of simplicity, are initiated using a single PBS script. In each R session, the same R code is used with a different input data file.

## PBS-based pipeline for array XCMS job

1. Peak-picking of individual data files in independent R sessions (1st PBS script *run-xcms-1.pbs* initiates an array job)
2. Merging generated XCMS objects into a single one and running the remaining XCMS steps in a single R session (2nd PBS script *run-xcms-2.pbs* initiates a serial job)

*run-xcms-1.pbs* script initiates 10 copies of the same job. Each of these subjobs run independently and are identical except for the value of the environment variable *PBS\_ARRAY\_INDEX*, which ranges from 1 to 10. Each copy is allocated with 1 core and 16GB of memory for 5 hours.*PBS\_ARRAY\_INDEX* is used as Rscript argument and specifies a single file from the list of 10. Another environment variable *PBS\_JOBNAME* encodes your given job name and is used to make a directory for job’s output. Variable *PBS\_JOBID* specifies the unique job ID, which is given by the PBS scheduler once a job is submitted to the queue. Job ID here is used to write a unique Rout file for easier comparison with other runs.

#!/bin/sh  
#PBS -N xcms\_v1  
#PBS -l walltime=05:00:00  
#PBS -l select=1:ncpus=1:mem=16gb  
#PBS -J 1-10  
dir="/work/username/scripts\_dir"  
file\_list="/work/username/filelist.txt"  
out\_dir="/work/username/output\_dir/$PBS\_JOBNAME"  
if [ ! -d "$out\_dir" ]; then  
 mkdir "$out\_dir"  
fi  
cd "$dir"  
module load intel-suite libxml2 hdf5/1.8.14-serial netcdf/4.4.1 R/3.4.0  
module load boost  
Rscript --no-save --no-restore --verbose xcms-1.R "$file\_list" "$PBS\_ARRAY\_INDEX"  
 "$out\_dir" > "$out\_dir/$PBS\_JOBID.Rout" 2>&1

The corresponding *xcms-1.R* script pick-peaks a single datafile and saves xcmSet object into an rds file with unique name in the same directory:

args = commandArgs(trailingOnly=TRUE)  
.libPaths("/apps/R/3.4.0/lib64/R/library")  
library(xcms)  
  
###--- Filelist  
file\_list <- paste0(args[1])  
files <- read.table(file = file\_list, stringsAsFactors = FALSE, header = F, sep = "\t")  
  
###--- Datafile to process  
f <- as.numeric(paste0(args[2]))  
file <- files[f,1]  
  
###--- Output dir  
output\_dir<- paste0(args[3])  
  
###--- Peak picking ----  
xset <- xcmsSet(files = file, ...)  
saveRDS(xset, file = paste0(output\_dir,"/xcms-1-", f, ".rds"))

*run-xcms-2.pbs* script initiates a serial job for remaining XCMS steps. It specifies the input directory where first jobs objects were generated, an output directory and the number of BPPARAM workers for fillPeaks() function (as many workers, as requested ncpus):

#!/bin/sh  
#PBS -N xcms\_v1  
#PBS -l walltime=10:00:00  
#PBS -l select=1:ncpus=24:mem=50gb  
input\_dir="/work/username/output\_dir/$PBS\_JOBNAME"  
out\_dir="$input\_dir"  
if [ ! -d "$input\_dir" ]; then  
 echo "no input\_dir, exit the job"  
 exit  
fi  
module load intel-suite libxml2 hdf5/1.8.14-serial netcdf/4.4.1 R/3.4.0  
module load boost  
Rscript --no-save --no-restore --verbose gset.R "$input\_dir" "$out\_dir" "$NCPUS"   
 > "$out\_dir/$PBS\_JOBID.Rout" 2>&1

*gset.R* script (note that graphic display is not supported on non-interactive jobs, thus plotting must be disabled):

args = commandArgs(trailingOnly=TRUE)  
.libPaths("/apps/R/3.4.0/lib64/R/library")  
library(xcms)  
  
###--- Input dir  
input\_dir <- paste0(args[1])  
  
###--- Output dir  
output\_dir<- paste0(args[2])  
  
###--- BPPARAM workers  
bw <- as.numeric(paste0(args[3]))  
  
###--- Load xcmsSet objects into one  
input <- list.files(input\_dir, pattern = ".rds", full.names = T)  
input\_l <- lapply(input, readRDS)  
  
xset <- input\_l[[1]]  
for(i in 2:length(input\_l)) {  
 set <- input\_l[[i]]  
 xset <- c(xset, set)  
 }  
  
###--- Grouping ----  
gset <- group(xset, ...)  
save(list = c("gset"), file = paste0(output\_dir,"/xcms-2.RData"))  
  
###--- RT correctionn ----  
rset <- retcor.peakgroups(gset, plottype = "none", ...)  
save(list = c("rset"), file = paste0(output\_dir,"/xcms-3.RData"))  
  
###--- Grouping no2 ----  
grset <- group(rset, ...)  
save(list = c("grset"), file = paste0(output\_dir,"/xcms-4.RData"))  
  
###--- Filling peaks ----  
fset <- fillPeaks(grset,  
 method="chrom",  
 BPPARAM = MulticoreParam(workers = bw))  
save(list = c("fset"), file = paste0(output\_dir,"/xcms-5.RData"))

Is possible to start a job on the condition that another one completes beforehand, where the input to one job is generated by the previous job in a pipeline. Job dependency is defined in PBS script using the -W flag.

XSET\_JOB\_ID=`qsub run-xset.pbs`  
qsub -W depend=afterok:$XSET\_JOB\_ID run-gset.pbs

# Running R for XCMS on AX4

If access to AX4 has been given to you, it could be an easier approach to using XCMS than the CX1, since a large amount of memory can be requested for a single R session. XCMS functions would be applied for all datafiles at the same time using multicore parallelisation supported by BiocParallel library.

## PBS script with check-points

The PBS scheduler on AX4 is prone to crashes, such as interruptions of the run and subsequent re-initiation. There is no way to save output of a session, which was terminated, but it is possible to re-iniate a code from the last check-point, rather than from the very start.

The check-points could be R objects, generated by individual XCMS functions throughout the workflow. To find the last check-point, PBS bash script would look for specifically named R objects in the output directory and initiate only one of the provided R scripts, depending on which R objects have already been generated. Different R scripts **start** the XCMS workflow at different stages. Each script saves one RData object:

1. *xcms-1.R* starts with xcmsSet()
2. *xcms-2.R* starts with group()
3. *xcms-3.R* starts with retcor()
4. *xcms-4.R* starts with second group()
5. *xcms-5.R* starts with fillPeaks()

#!/bin/sh  
#PBS -N xcms\_v1  
#PBS -l walltime=10:00:00  
#PBS -l select=1:ncpus=40:mem=200gb  
file\_list="/work/username/filelist.txt"  
out\_dir="/work/username/out\_dir/$PBS\_JOBNAME"  
check\_1="$out\_dir/xcms-1.RData"  
check\_2="$out\_dir/xcms-2.RData"  
check\_3="$out\_dir/xcms-3.RData"  
check\_4="$out\_dir/xcms-4.RData"  
check\_5="$out\_dir/xcms-5.RData"  
if [ ! -d "$out\_dir" ]; then  
 mkdir "$out\_dir"  
fi  
module load intel-suite libxml2 hdf5/1.8.14-serial netcdf/4.4.1 R/3.4.0  
module load boost  
if [ -e "$check\_5" ]; then  
 echo "run is over"  
else  
 if [ -e "$check\_4" ]; then  
 Rscript --no-save --no-restore --verbose xcms-5.R "$NCPUS" "$file\_list" "$out\_dir"  
 "$check\_4" > "$out\_dir/$PBS\_JOBNAME.Rout" 2>&1  
 else  
 if [ -e "$check\_3" ]; then  
 Rscript --no-save --no-restore --verbose xcms-4.R "$NCPUS"  
 "$file\_list" "$out\_dir" "$check\_3" > "$out\_dir/$PBS\_JOBNAME.Rout" 2>&1  
 else  
 if [ -e "$check\_2" ]; then  
 Rscript --no-save --no-restore --verbose xcms-3.R "$NCPUS" "$file\_list"  
 "$out\_dir" "$check\_2" > "$out\_dir/$PBS\_JOBNAME.Rout" 2>&1  
 else  
 if [ -e "$check\_1" ]; then  
 Rscript --no-save --no-restore --verbose xcms-2.R "$NCPUS"  
 "$file\_list" "$out\_dir" "$check\_1" > "$out\_dir/$PBS\_JOBNAME.Rout" 2>&1  
 else  
 Rscript --no-save --no-restore --verbose xcms-1.R "$NCPUS"  
 "$file\_list" "$out\_dir" > "$out\_dir/$PBS\_JOBNAME.Rout" 2>&1  
 fi  
 fi  
 fi  
 fi  
fi

# Useful links

A lot of information provided here was sourced from: