Practical guide to running R on HPC

Elzbieta Lauzikaite

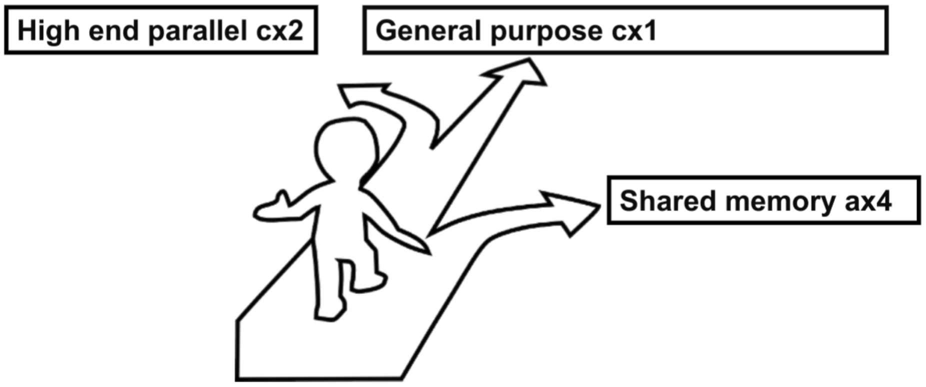
5/3/2018

Contents

1 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	CX2
For large number of small jobs	For very large datasets	For numerically intensive jobs
Ideally jobs should fit a single node	Consists of two systems: 160 and 1280 cores each	Reserved for parallel jobs
Total ~1700 nodes and ~ 33000 cores	Total 16TB RAM and 1400 cores	Total ~19000 cores

1.1 CX1

Table 2: Jobs allowed on CX1 system.

Job class	Number of nodes N	ncpus node	Max mem node	Max walltime hr	Max number of running jobs per user
throughput	1	1-8	96GB	up to 72hr	unlimited for jobs <=24hr in length
general	1 - 16	16	62GB or 124GB	up to 72hr	unlimited for jobs <=24hr in length
singlenode	1	24	124GB	up to 24hr	10
multinode	2 - 16	12	46GB	up to 48hr	unlimited
debug	1	1-8	96GB	up to 30 mins	1
large memory	1	12	190 or 250GB	48 hr	unlimited
GPU	1-1	1-4	16GB	48hr	8
long	1	1-8	96GB	72 - 1000 hr	1

1.2 AX4

Table 3: Jobs allowed on AX4 system.

Job class	Number of nodes N	ncpus node	Max mem node	Max walltime hr
long	1	1-7	127	100
large	1	8-600	8000	100
monster (will not be run)	NA	NA	NA	NA

2 Set up

2.1 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 **self-service portal**. This will give you access to CX1 free-of-charge, for usage of AX4/CX2, contact HPC/RCS support team.

2.2 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 Putty.

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.

2.3 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.

2.3.1 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 file systems are not appropriate for the storage of sensitive data.

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
```

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

2.3.2 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.

Table 4: Storage available to every user.

Storage	Purpose
$HOME	25GB of space which can be extended to 100GB
	Intended for storing binaries and source only
	/home/username
$WORK	Main work directory on CX1. 250GB of spcace which can be extended to 1TB
	Intended for storing datafiles and long term storage
	/work/username
$TMPDIR	Temporal directory created for CX1 running jobs
	Is deleted when the job finishes
	/tmp/*
$SCRATCH	High performance $WORK filesystem for CX2/AX4
	Intended for long-term data storage. X2/AX4 jobs should write directly to it
	/scratch*/username

Table 5: External storage options.

Storage	Purpose
H:drive	Access from CX1 only
	Up to 8GB free of charge
	For non-sensitive data only
	Run 'module load hdrive' and 'hdrive'
BOX	Access from CX1 only
	Unlimited data storage with 15GB per individual datafile
	For non-sensitive data only
	Run 'module load box' and 'box'
Network drives	Access from CX1/AX4
	E.g. MED-BIO
	Access granted to group members only

3 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 Imperial website.

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 HPC help desk.

```
.libPaths("/home/username/R/x86_64-pc-linux-gnu-library/3.4", .libPaths())
library(xcms)
```

4 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 -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
```

4.1 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 Imperial HPC Wiki page:

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

4.2 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])
```

5 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.

5.1 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])
```

```r
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):

```sh
#!/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):

```r
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
```

6 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.

6.1 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
```

7 Useful links

A lot of information provided here was sourced from:

1. Imperial HPC Wiki page
2. Imperial HPC course slides
3. Imperial website
4. Introduction to using R on HPC
5. PBS scheduler environment variables