Shark simulation tutorial using R

Jeffrey Durieux, Elise Dusseldorp, Wouter van Loon & Juan Claramunt

07/09/2018

Contents

1 Introduction

In this document, we describe the set-up of a simulation study and the performance of it on a cluster computer (SHARK). Also, we introduce basic knowledge on how to use the SHARK cluster.

An example is included to illustrate each step of the simulation study and its computation using a cluster computer.

The document is structured as follows:

- Chapter 1: How to connect to the SHARK cluster;
- Chapter 2: File management on a cluster;
- Chapter 3: Specification of a simulation study;

- Chapter 4: Preparation of a simulation study;
- Chapter 5: Performance of a simulation study on a cluster computer;
- Chapter 6: Collecting the results from the cluster computer in one large matrix and evaluation of the results by ANOVA;
- Chapter 7: Email notifications;
- Chapter 8: Installing R packages on Shark;
- Chapter 9: Warning for writing shell scripts on Windows;
- Chapter 10: Useful Linux commands.

Note: *You may find all functions required for the example in the MS-Github page click here*

2 How to connect to the SHARK cluster

There are several options to connect to SHARK. They depend on your operating system (Windows, Linux, Mac OS, etc.).

2.1 Connection using a device that is connected to the FSW network by cable (not wifi).

Details of the connection:

Research Jump server:

IP address: 145.88.65.151
Hostname: shark.researchlumc.nl
Port: 22

2.1.1 Windows users

In order to connect to a cluster computer using Windows it is necessary to install a client SSH software. The most common options are Putty and MobaXterm (recommended option).

Connect with Putty

```
1. Open putty.
2. Introduce IP address or hostname (see details above).
3. Select ssh.
4. (Optional) Set a name under saved sessions and save.
5. Left click on open.
6. Introduce your SHARKusername.
7. Introduce your password.
8. Congratulations! You are now in SHARK.
```

Connect with MobaXterm (recommended)

First option (it saves the session is saved for future connections):

```
1. Open MobaXterm.
2. Click on sessions.
3. Click on ssh.
4. Fill in the 'Remote host' (shark.researchlumc.nl).
5. Select specify username and introduce your SHARKusername, and set port to 22.
6. (Optional) Check the 'Follow SSH path' in the 'Advanced SSH settings'-TAB.
7. (Optional) Un-check the 'Display reconnection message at session end' in
```

the 'Bookmark settings'-TAB.
8. Click on OK.
9. Introduce your password.
10. Congratulations! You are now in SHARK.

Second option (faster):

1. Open MobaXterm
2. Click on start local terminal.
3. Type: ssh <SHARKusername>@145.88.65.151 .
4. Introduce your password.
5. Congratulations! You are now in SHARK.

2.1.2 Linux users

1. Open the terminal.
2. Type: ssh -XY <SHARKusername>@shark.researchlumc.nl
3. Type your password.
4. Congratulations! You are now in SHARK.

2.1.3 Mac users

1. Open the terminal.
2. Type: ssh -XY <SHARKusername>@shark.researchlumc.nl
3. Type your password.
4. Congratulations! You are now in SHARK.

2.2 Connection using any device.

For any device on any location, follow the next steps.

Details of the connection:

Research Jump server:

IP address: 145.88.35.10
Hostname: res-ssh-alg01.researchlumc.nl
Port: 22

2.2.1 Windows users

Connect with Putty

1. Open putty.
2. Introduce IP address or hostname (see details above).
3. Select ssh.
4. (Optional) Set a name under saved sessions and save.
5. Left click on open.
6. Introduce your SHARKusername.
7. Introduce your password.
8. Type: ssh shark .
9. Introduce your password again.
10. Congratulations! You are now in SHARK.

Connect with MobaXterm (recommended)

First option (it saves the session is saved for future connections):

```
1. Open MobaXterm.
2. Click on sessions.
3. Click on ssh.
4. Type 'shark' in 'Remote host'.
5. Select specify username and introduce your SHARKusername, and set port to 22.
6. (Optional) Check the 'Follow SSH path' in the 'Advanced SSH settings'-TAB.
7. Check the 'Connect through SSH gateway (jump host)' in the 'Network settings'-TAB.
8. Fill the gateway SSH server (145.88.35.10), the port (22) and the user (SHARKusername)
in the 'Network settings'-TAB.
9. (Optional) Un-check the 'Display reconnection message at session end' in
the 'Bookmark settings'-TAB.
10. Click on OK.
11. Introduce your password.
12. Once in the terminal, introduce your password again.
13. Congratulations! You are now in SHARK.
```

Second option (faster)

```
1. Open MobaXterm
2. Click on start local terminal.
3. Type: ssh SHARKusername@145.88.35.10 .
4. Introduce your password.
5. Type: ssh shark .
6. Introduce your password again.
7. Congratulations! You are now in SHARK.
```

2.2.2 Linux users

Recommended option by the mantainers of SHARK:

Before connecting the first time follow the next steps:

```
1. Open a Terminal and create a directory (if needed) .ssh in your home directory.
2. Create a config file inside your .ssh directory and add the following: vi ~/.ssh/config
3. Add also:
  Host resshark
    Compression yes
    CompressionLevel 9
    ForwardX11 yes
    Hostname 145.88.65.151
    Localforward 6001 145.88.65.151:22
    User SHARKUsername
    ProxyCommand ssh -q -X -C SHARKUsername@145.88.35.10 nc %h %p
4. Set the permissions for the user for the .ssh directory: chmod 700 ~/.ssh
5. Set the permissions for the user on all the files in your .ssh directory: chmod 600 ~/.ssh/*
```

Once the previous steps are finished, follow the next ones to connect:

```
1. Type ssh resshark
2. Introduce your password.
3. Introduce your password again.
4. Congratulations! You are now in SHARK.
```

Faster option (does not require the config file):

```
1. Open a Terminal.
2. Type ssh -X SHARKusername@res-ssh-alg01.researchlumc.nl
3. Type ssh shark.
4. Congratulations! You are now in SHARK.
```

2.2.3 Mac users

Recommended option by the mantainers of SHARK.

Before connecting the first time follow the following steps:

```
1. Open a Terminal and create a directory (if needed) .ssh in your home directory.
2. Create a config file inside your .ssh directory and add the following: vi ~/.ssh/config
3. Add also:
  Host resshark
    Compression yes
    CompressionLevel 9
    ForwardX11 yes
    Hostname 145.88.65.151
    Localforward 6001 145.88.65.151:22
    User SHARKUsername
    ProxyCommand ssh -q -X -C SHARKUsername@145.88.35.10 nc %h %p
4. Set the permissions for the user for the .ssh directory: chmod 700 ~/.ssh
5. Set the permissions for the user on all the files in your .ssh directory: chmod 600 ~/.ssh/*
```

Once the previous steps are finished, follow the next ones to connect:

```
1. Type ssh resshark
2. Introduce your password.
3. Introduce your password again.
4. Congratulations! You are now in SHARK.
```

Fast option (does not require the config file):

```
1. Open a Terminal.
2. Type ssh -Y SHARKusername@res-ssh-alg01.researchlumc.nl
3. Type ssh shark.
4. Congratulations! You are now in SHARK.
```

You can find more information here.

3 File management in a cluster

The file management (copy, move, upload, download files/folders) also depends on your operating system.

3.1 Windows users

Windows users need a SSH File Transfer Protocol (SFTP) client software. MobaXterm (recommended SSH client software) includes a SFTP, however, Putty does not. Therefore, in case you use Putty, other software such as Filezilla or WinSCP is needed.

3.1.1 MobaXterm users

The SFTP in **MobaXterm** can be found in the TABs on the left side (see Figure 1). In order to use it, you must connect to SHARK first. Once in SHARK, the SFTP tab contains icons to:

- Move to the parent directory
- Download files
- Upload files
- Refresh the folder
- Create a new directory
- Create a new file
- Delete a new file

Below these icons there is the path corresponding to your current folder (directory) in SHARK.

Below the path you can find the files and folders in the current path.

3.1.2 Putty users

If you are using **Putty** to connect to SHARK, the process is far more difficult. It is necessary to download a software such as Filezilla or WinSCP and create an SSH tunnel (in case you are not connected to the FSW network by cable). Unluckily, creating these tunnels is not possible with Filezilla.

Using cable connection:

First **Filezilla** is installed. Fill the host (145.88.65.151), the username (SHARKusername), the password (SHARKpassword) and the port (22) as in Figure 2, and press the ENTER key.

On the left side of the screen you will have the files/folders on your computer and on the right side of the screen the files/folders on the SHARK cluster. You can manage your files in a similar way as in the Windows environment.

Not using cable connection:

First **WinSCP** is installed. Click on New Session. Go to advanced settings and select tunnel on the left side (see Figure 3). Select "connect through SSH tunnel". Fill the hostname (145.88.35.10), Port number (22), user name (SHARKusername) and password (SHARKpassword) and click on OK. Then, fill in the new window the hostname (145.88.65.151), Port number (22), user name (SHARKusername) and password (SHARKpassword). Once connected, it works similarly to Filezilla and Windows.

NOTE: It is important to realize that the first hostname is not the same as the second host name!

3.2 Linux and Mac users

Open the terminal and use the command **scp** to transfer files to/from the cluster.

The basic syntax of scp is:

scp [from] [to]

where "from" can be a filename or a folder and "to" contains your netid, the hostname of the cluster login node, and the destination directory if you are tranfering files to SHARK and vice versa if you are downloading files from SHARK. For example, to upload files:

scp myfile.txt SHARKusername@shark.researchlumc.nl:/exports/fsw

In this example, "myfile.txt' is copied to the directory /exports/fsw of SHARK.

If you want to transfer the whole folder with its content you need to add -r, for example:

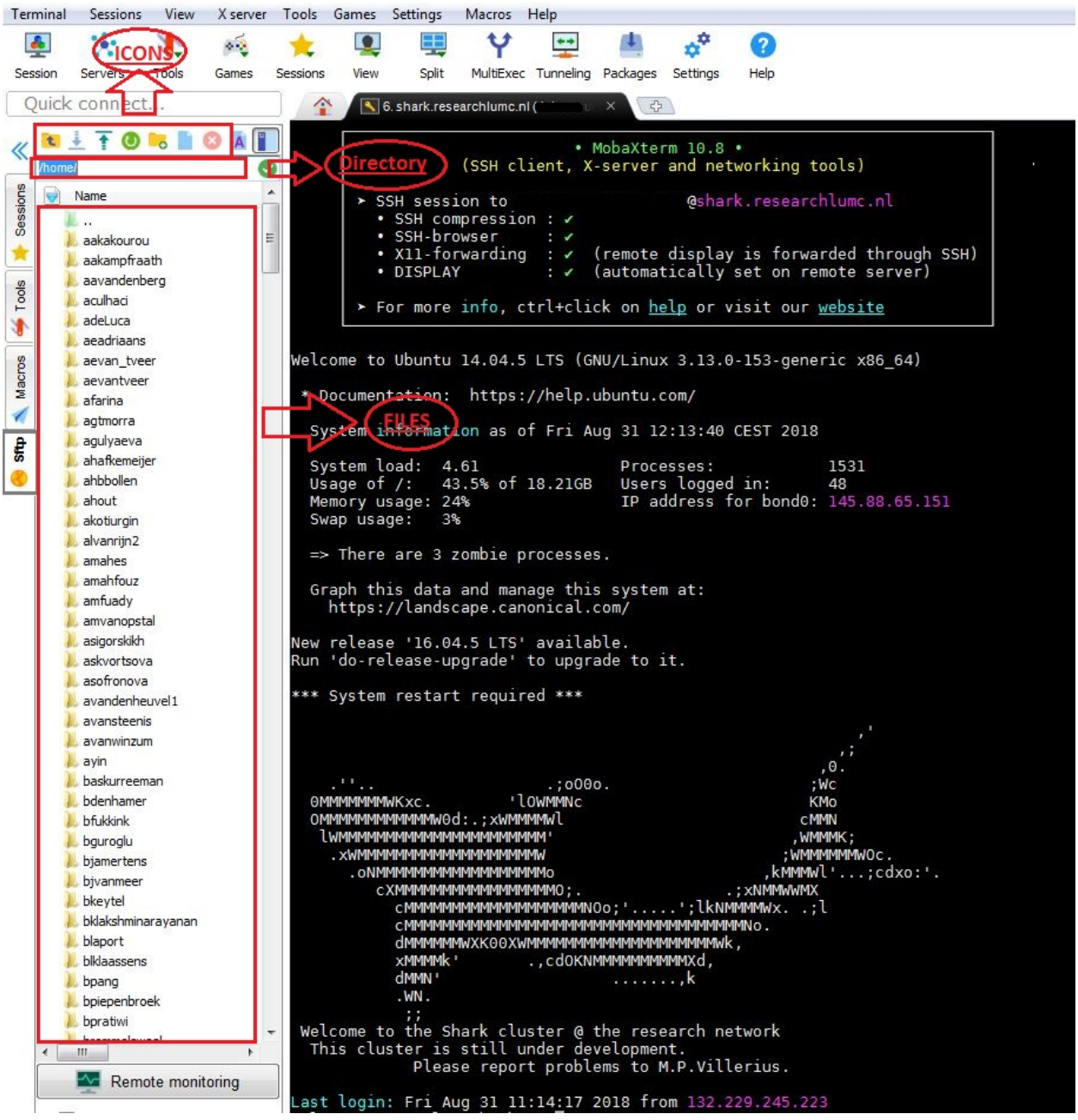

Figure 1: MobaXterm SFTP

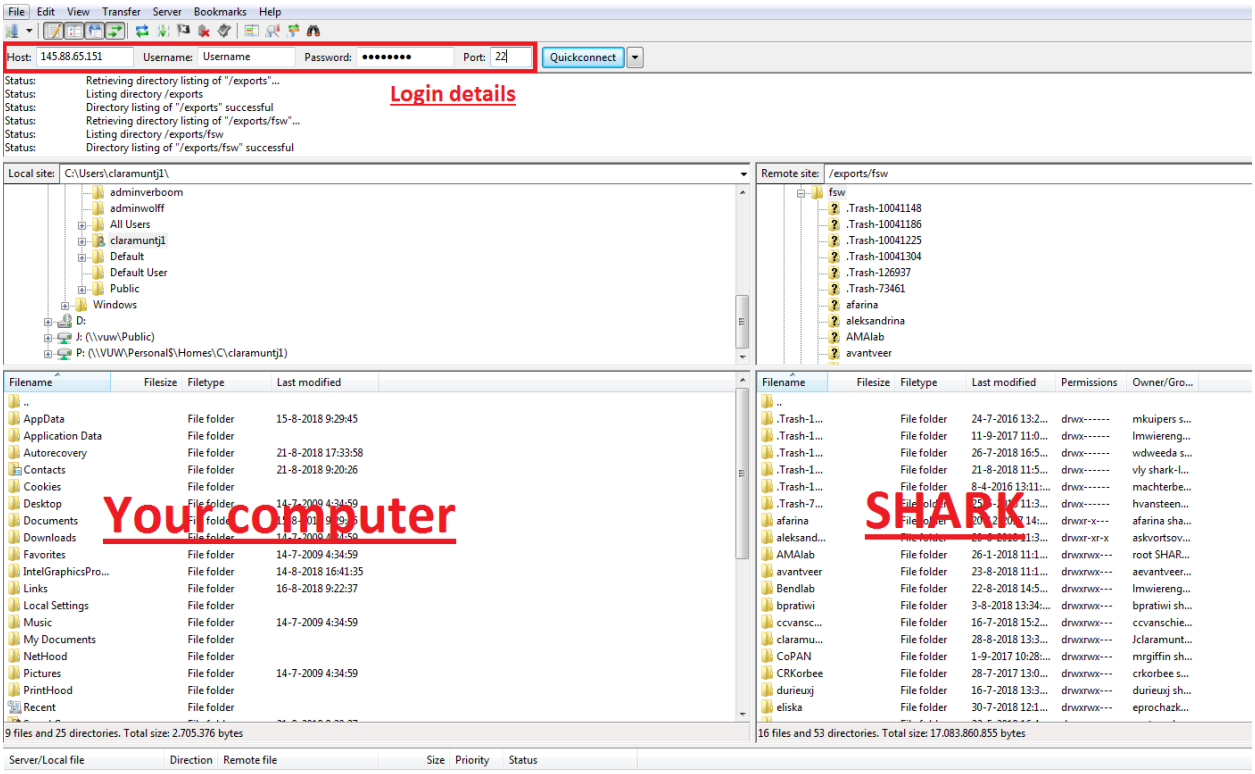

Figure 2: Filezilla

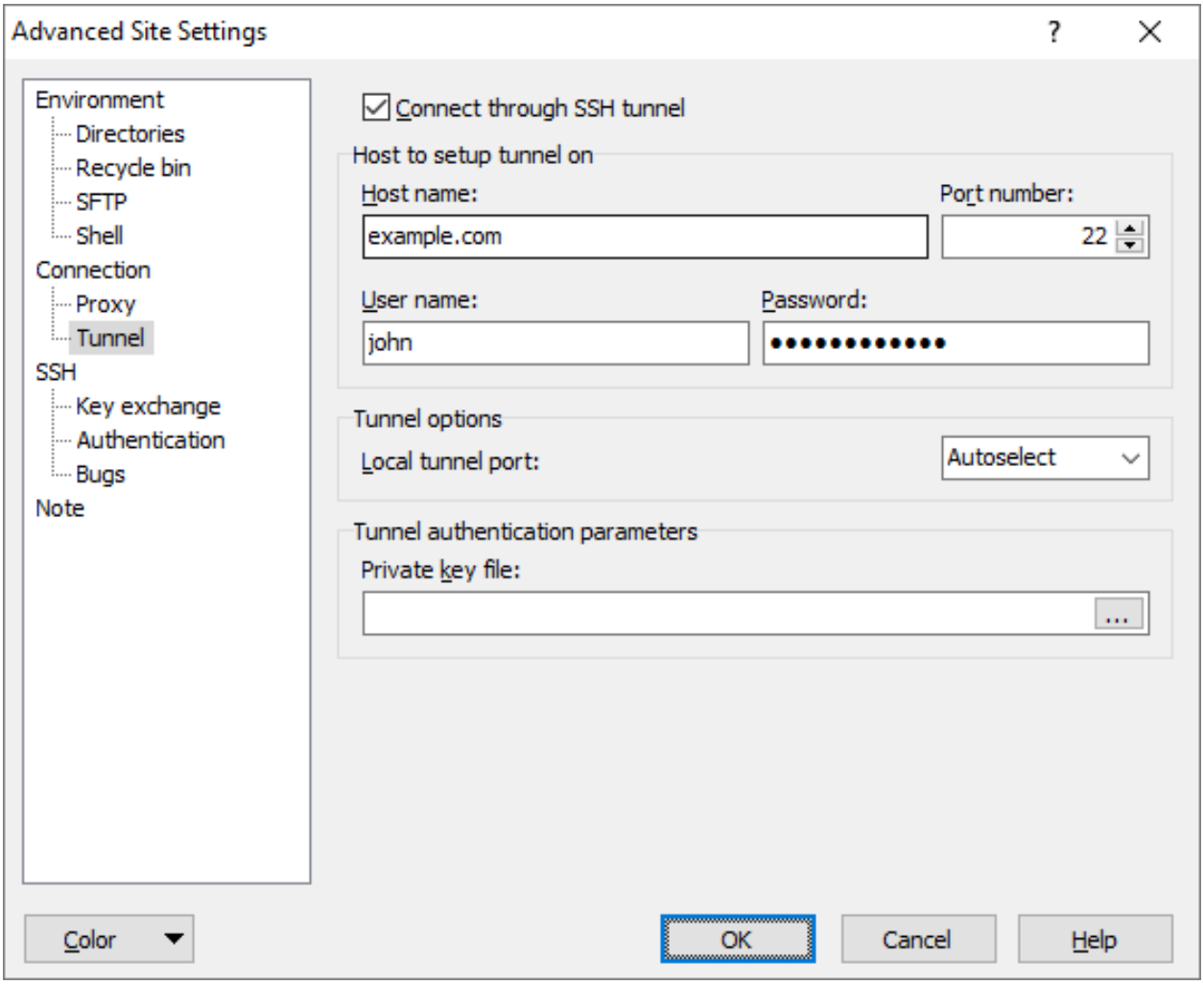

Figure 3: WinSCP advance options window

scp -r mydirectory SHARKusername@shark.researchlumc.nl:/exports/fsw

To download files/folder from SHARK just interchange the paths:

scp SHARKusername@shark.researchlumc.nl:/exports/fsw/myfile.txt MyDocuments

In this example, myfile.txt is downloaded from SHARK to your local folder MyDocuments.

You can also transfer files between your local computer and a cluster using an SFTP client software, such as Cyberduck (OSX) or FileZilla (Linux).

For more information on how to transfer files from or to a cluster, *click here*.

4 Specification of a simulation study

In a full factorial simulation study, we consider the following 4 steps:

- Step 1: Research Question
- Step 2: Gauge and Design
- Step 3: Statistical Analysis
- Step 4: Evaluation Criteria

4.1 Step 1: Research Question

First, you formulate the research question you would like to answer by means of a simulation study.

In our example, the research questions are: Does the Mann-Whitney U test show higher power (i.e., lower Type II error) than the independent samples t-test? And does the difference in power between the two methods depend on the total sample size and the mean difference between the two groups?

We hypothesize that the power of the Mann-Whitney test is higher than the independent samples t-test, especially in smaller samples.

4.2 Step 2: Gauge and design

Start with the formulation of:

- The true scenario(s)/model(s) from which you will generate your artificial data, and specify from which distributions you generate the variables that are included in the model.

  -In our example, we have a continuous, standardized outcome variable and one variable group that has two categories (i.e., two groups).

- Specify the design factors (i.e. the factors you are going to vary systematically).

  In our example, there are two design factors:

  – Samp (sample size, having 4 levels: N = 10, 20, 40, and 80 in the example);

  – Es (effect size, having 3 levels: d = 0.2, 0.5 and 0.8 in the example). In this example, the effect size is the standardized difference in means between the two groups.

- Number of replications within each cell: *k*. In our small example, k = 10.

Then, our design contains in total 4 (Samp) by 3 (Es) 12 cells, and for each cell we will generate k = 10 data sets. Thus, in total, we analyze 120 data sets.

4.3 Step 3: Statistical Analysis

Specify the type of analysis you want to perform for each dataset (of each cell of the design). Often you want to compare two types of analysis methods (a new method compared to an old method).

The two methods used in the statistical analysis of the example are the t-test and the Mann-Whitney test which can be found in the *Method_old.R* and *Method_new.R* files respectively.

4.4 Step 4: Evaluation Criteria

Specify the evaluation criteria which you want to use to compare the two methods.

In our example, the evaluation criterion used to compare both methods (the t-test and the Mann-Whitney) is the number of times that the test is significant using $\alpha = 0.05$. If it is significant it means that the test correcly rejects the null hypothesis of no difference in means between the two groups.

5 Preparation of a simulation study

Write the following functions in `R`:

- A function that generates one data set for each true scenario in the design. The input arguments of this function are the type of true model and the levels of the factors of the simulation design. In our example, we refer to this function as *MyDataGeneration.R*. The output of this function is a simulated data set (SimData);
- One or two functions that analyse the data according to one or two methods. In our example, we refer to these functions as *Method_old* and *Method_new.R*. The input argument of these functions is a dataset (and possibly more input arguments). The output of this function contains the relevant results. In our example, we save the output of these functions as MyAnalysisResult (see *MainSimulationScript.R* below);
- A function that applies the evaluation criteria. In our example, we refer to this function as *MyEvaluation.R*. The input of this function is MyAnalysisResult, this is, the output of the methods.

6 Performance of a simulation study on a cluster computer

6.1 The main simulation script:

Below you can see the main simulation script called *MainSimulationScript.R*. This is the script that contains your whole simulation study. Here, your design is coded, analyses are specified and evaluation criteria are computed. Moreover, saving your output to your directory is specified.

Some usefull and sometimes necessary functions:

- commandArgs()
  - captures supplied command line arguments (see section 6.2 and 6.3)
- expand.grid()
  - usefull function that creates a data frame from all combinations of factor variables
- set.seed()
  - important for replication of your simulation study.
- do.call()
  - neat function to execute a function with a list of supplied arguments

```
#!/bin/bash
# MainSimulationScript.R

# args is a stringvector
args <- commandArgs(TRUE)
args <- as.numeric(args)

RowOfDesign <- args[1]
Replication <- args[2]

### Simulation design example
samp <- c(10, 20, 40, 80)
es <- c(0.2, 0.5, 0.8)

# Design is a data.frame with all possible combinations of the factor levels
# Each row of the design matrix represents a cell of your simulation design

Design <- expand.grid(samp = samp, es = es)

# set a random number seed to be able to replicate the result exactly
set.seed((Replication + 1000)*RowOfDesign)

###Preparation:
##Install R packages first in separate job file  (see chapter 9)
#Always use library() herento activate the non-standard R-packages, for example
#library(ica)
#By the way: this is just to illustrate, we do not use this package for our example

source("/home/Jclaramuntganzalez/sharktutorial/MyDataGeneration.R")
source("/home/Jclaramuntganzalez/sharktutorial/Method_new.R")
source("/home/Jclaramuntganzalez/sharktutorial/Method_old.R")
source("/home/Jclaramuntganzalez/sharktutorial/MyEvaluation.R")

#An alternative to the previous lines is to create your own directory
#and upload the files to it. For example:
#source("/yourdirectorypath/MyDataGeneration.R")
#source("/yourdirectorypath/Method_new.R")
#source("/yourdirectorypath/Method_old.R")
#source("/yourdirectorypath/MyEvaluation.R")

########## simulation ############
# Generate data
SimData <- do.call(MyDataGeneration, Design[RowOfDesign, ]  )

# Analyze data set with Method_new
tmp <- proc.time()
MyAnalysisResult1 <- Method_new(SimData)

#Analyze data set with Method_old
MyAnalysisResult2 <- Method_old(SimData)
time <- proc.time() - tmp #save the time to run the analyses of one cell of the design.
# Also possible to time both analyses seperately

```

```r
#Combine relevant results of the analysis by the two methods in a vector (optional)
MyAnalysisResult <- c(MyAnalysisResult1$p.value, MyAnalysisResult2$p.value)

#Evaluate the analysis results of Method_new and Mehod_old
MyResult1 <- MyEvaluation(MyAnalysisResult1)
MyResult2 <- MyEvaluation(MyAnalysisResult2)

#combine the results in a vector:
MyResult <- c(MyResult1, MyResult2)

# save stuff on export
setwd("/exports/fsw/claramuntj/sharktutorial")
#Alternatively, you can also create your folder in the export/fsw directory
#and set it as your working directory.


# Write output (also possible to first save everything in a list object)

#optional to save the data
save(SimData, file =paste("Simdata","Row", RowOfDesign,
      "Rep", Replication ,".Rdata" , sep =""))

#optional to save the full analysis result
save(MyAnalysisResult, file =paste("Analysis","Row", RowOfDesign,
      "Rep", Replication ,".Rdata" , sep =""))

save(MyResult, file =paste("MyResult", "Row", RowOfDesign,
      "Rep", Replication ,".Rdata" , sep =""))

#optional to save timing of analyses
save(time, file =paste("Time", "Row", RowOfDesign,
      "Rep", Replication ,".Rdata" , sep =""))
```

6.2 The bash script used to run Jobs

Below you can find a bash script (let's call it *RunMySimulation.sh*) that is used on Shark to run an R-script The bash script 'starts' the R-script *'MainSimulationScript.R'* with the command R CMD BATCH and includes some provided arguments: $i and $j. The $ sign indicates that it is a replacement character, with a for loop you can change those values on the fly (see part 6.3). The values provided to these replacement characeters will be collected with the commandArgs() function in our R-script.

```
#!/bin/bash
#$ -S /bin/bash
#$ -q all.q
#$ -N sim_1
#$ -l h_vmem=1G
#$ -cwd
#$ -j Y
#$ -V


R CMD BATCH "--args $i $j" MainSimulationScript.R
```

For the small simulation example we are going to run, requesting 1G of memory (-l h_vmem=1G) is enough.

6.3 Start your embarrassingly parallel jobs

We have an R-script with simulation code, *MainSimulationScript.R*. We also have a bash script *RunMySimulation.sh* which 'starts' the R-script. But how do we actually start our simulation? We simply use the *qsub* command in a double for-loop. Thus, our jobs are submitted to shark in a parallel fashion.

Remember, for the current example the simulation design consists of two factors: Sample size (Samp) and effect size (Es). The factors have 4 and 3 levels respectively. So we have a 4x3 factorial design. And we want to replicate this design $k = 10$ times. Thus, we will generate, analyze and evaluate a total of 4 x 3 x 10 = 120 datasets. This can be done as follow:

```
for replication in {1..10}; do for rownumber in {1..12}; do
qsub -v i=$rownumber,j=$replication RunMySimulation.sh; done; done
```

Note: *In order to use R on shark, you first need to 'load' it. This can simply be done on the command line by typing module load R/3.5.0 (or select another R version available on SHARK)*

You can submit the double for loop on the command line in Shark. This double for loop will start 10 x 12 = 120 jobs that will run in parallel. Each job has no dependency on each other, meaning that you can run each row of the design completely seperate. This is known as embarrasingly parallel computing and it saves you a lot of computation time!

6.3.1 Job Arrays (alternative to the for-loop)

A downside of using for-loops to submit jobs is that each job is submitted to the cluster individually. This is taxing for the head node, especially if the number of jobs is large. An alternative to the use of the for-loop for submitting jobs to SHARK is to use a job array.

A job array is a special job which consist of similar tasks that you want the cluster to execute. The primary advantage of using a job array is that the entire set of tasks is submitted to the queue as a single entity. The array as a whole is assigned a job identifier (column 'job-ID' in the output of `qstat`), and each task within the array is assigned an individual task identifier (column 'ja-task-ID'). In the `qstat` output, waiting tasks are collapsed into a single row.

For example, if we have 1000 tasks queued this will appear as a single job with ja-task-ID 1-1000 (i.e. "tasks 1 through 1000 are waiting"), rather than as 1000 individual jobs. Running tasks are still displayed individually, so you can see on what node they are being executed, etc. The presence of both a job and task identifier makes it easy to request information on, or make changes to, both the individual tasks and the array as a whole.

Creating a job array is very simple: just include the '-t' flag in your bash script, for example, the script *'RunMySimulation.sh'* generating *'RunMySimulationJobArray.sh'*. An example on how to create a job array is as follows:

```
#!/bin/bash
#$ -q all.q
#$ -N my_array
#$ -V
#$ -cwd
#$ -j y
#$ -l h_vmem=600M
#$ -t 1-10
i=1
j=$SGE_TASK_ID
```

```
R CMD BATCH "--args $i $j" MainSimulationScript.R
```

Submitting this shell script to the cluster will create a job array called my_array, which consists of 10 tasks (-t 1-10). Each task executes the file 'MainSimulationScript.R'. In this small example we are running the 10 repetitions of the first combination of parameters in the design.

It can be useful to save not only the results of a script, but also what is printed in the R console during execution. When using the R CMD BATCH command, you can easily do this by specifying a .Rout file. For example:

```
#!/bin/bash
#$ -q all.q
#$ -N my_array
#$ -V
#$ -cwd
#$ -j y
#$ -l h_vmem=600M
#$ -t 1-40

module load R/3.5.0

R CMD BATCH --no-save --no-restore my_script.R my_task_${SGE_TASK_ID}.Rout
```

This job array will save the R console output of each task to a .Rout file, which is a simple text file that can be read using e.g. `cat`.

In general, when running a job array, we do not want each task to do exactly the same thing: we want to vary experimental parameters, etc. Fortunately, the identifier for each task is stored in an environment variable called *SGE_TASK_ID* (as we can observe in the previous example). We can obtain the value of this variable in R by using e.g. `as.numeric(Sys.getenv("SGE_TASK_ID"))`, as in the following example:

```
# Experimental factors
f1 <- c(10, 20)
f2 <- c(0.2, 0.5)

# Replications
rep <- 1:10

# Matrix containing parameters for all individual tasks to perform
Design <- expand.grid(rep, f1, f2)

# Generate seeds
set.seed(123)
seed.list <- sample(.Machine$integer.max/2, size = nrow(Design))

# Obtain the Task ID
TID <- as.numeric(Sys.getenv("SGE_TASK_ID"))

# Apply some function to obtain some result
set.seed(seed.list[TID])
result <- my_function(Design[TID,])

# Save the result
save(result, file=paste0("<path_to_folder>/my_result_", TID, ".RData"))
```

Submitting a job array as the one in the example will now execute this function 10 times, each with a unique

combination of f1, f2 and seed.

7 Collecting the results

After all submitted jobs are done, you should collect all results from your output folder. Below you can find an R script (*ResultsCollecting.R*) that collects the right output and automatically stores the results in a matrix. This matrix can be used for further analyses such as an ANOVA.

Note that collecting the results can be done on Shark. However, sometimes it is easier to first download the results to your own device and then collect the final results.

Note that this script uses the package gtools. You first need to install this package.

```
# Collect all results and put it in a matrix

# set directory with results
setwd("~/Desktop/shark/Data/")

library(gtools)

## original design

samp <- c(10, 20, 40, 80)
es <- c(0.2, 0.5, 0.8)
Design <- expand.grid(samp = samp, es = es)

# pattern matching with grep
files <- dir()
idx <- grep(pattern = "MyResult", files)
files <- files[idx]

# test to see mixedsort() does it's job
scram <- sample(1:120, 120, replace = F)
files <- files[scram]

files <- mixedsort(files)

###
RepIdxList <- list()
for(rep in 1:10){
  RepIdxList[[rep]] <- grep(pattern = paste("Rep",rep,sep = "" ), files )
}

idx_to_remove <- RepIdxList[[1]] %in% RepIdxList[[10]]
RepIdxList[[1]] <- RepIdxList[[1]][!idx_to_remove]

idx <- unlist(RepIdxList)

files <- files[idx]

# rbind design k times and cbind files names
#(so that you can check whether the right values are taken)

Results <- do.call(what = rbind, args = replicate(10, Design, simplify = F))
```

```
Results <- cbind(Results, files)

# the sapply function loads the results and preserves the file name (in the rows).
# This is usefull for checking whether you did everything right

Res <- t( sapply(files, function(x) get(load(x) ) ) )

Results <- cbind(Results, Res)
colnames(Results) <- c("samp", "es", "files", "res1", "res2")

save(Results, file = "AllResults.Rdata")
```

8 Email notifications

When submitting a job you can use the -M and -m flags to send yourself email notifications. The -M flag is used to specify your email adress, while the -m flag is used to specify when you want to be notified: at the beginning of the job (b) and/or at the end (e), when the job is aborted (a) or suspended (s), or no notifications at all (n, the default).

You will notice that the job array examples above do not include email notifications: this is because the -m flag applies to each task individually. It is generally not recommended to use begin- and end-notifications with large job arrays, otherwise you will get an email at the start and end of each individual task, which is not fun for both the mail server and your inbox (the same applies for the for-loop). Unfortunately, SGE/OGS does not seem to support options for controlling notifications for the array as a whole. However, there are workarounds if you really require an email notification when all tasks are complete. For example, you could submit a simple job which waits in the queue until the entire array is finished, then executes and sends you an email, e.g.:

```
qsub -o /dev/null -e /dev/null -M YOUREMAIL@fsw.leidenuniv.nl -m ea -b y -l

h_rt=00:00:15 -hold_jid YOUR_ARRAY_NAME -N 'array complete' true
```

9 Installing R packages on Shark

Option 1:

Start an individual R session on shark:

```
$ qlogin.
$ module load R/3.5.0 (or another version)
$ R
```

Once R is open install the package you want as usual in R, using: install.packages(“”).

Choose a CRAN mirror, and say that you want to create a personal folder to install the package when Shark asks you (this will happen the first time you install a package).

Option 2 (only works when you already have a personal folder):

The first step is to create a personal folder (if you do not have one yet). To do so, type on the command line:

mkdir /home/YOUR_DIRECTORY/R/x86_64-pc-linux-gnu-library/3.5 (or the corresponding R version)

Then, create the R script to install the package, ‘InstallPackages.R’:

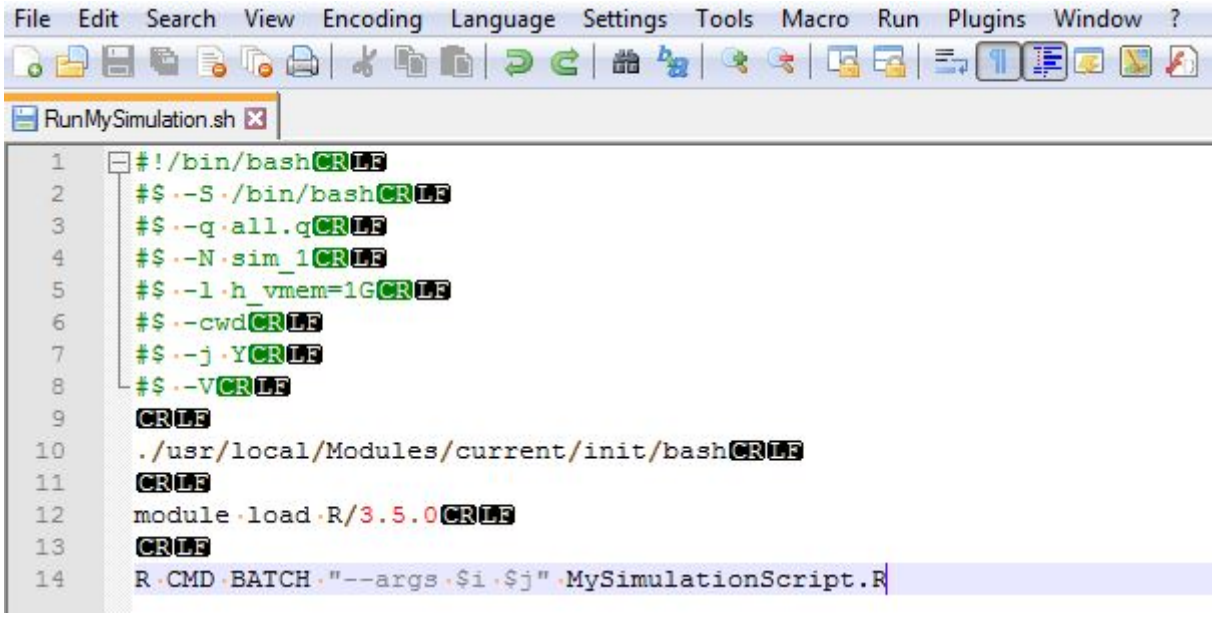

Figure 4: Windows line endings

```
install.packages(c("ica","Matrix"), Sys.getenv("R_LIBS_USER"), repos = "http://cran.case.edu")
```

Finally, generate the bash script 'InstallPackages.sh' to run the R script in SHARK.

```
#!/bin/bash
#$ -S /bin/bash
#$ -q all.q
#$ -N installpackages
#$ -cwd
#$ -j Y
#$ -V
#$ -m be
#$ -M YOUREMAIL@fsw.leidenuniv.nl

R CMD BATCH InstallPackages.R
```

How to run it in Shark: On the commandline type qsub InstallPackages.sh

Don't forget to load R first (module load R/3.5.0)

10 Warning for writing shell scripts on Windows: line endings

If you are a Microsoft Windows user and are writing shell scripts (.sh) for execution on Shark, make sure that your files have the correct Unix-style line endings, rather than the default Windows-style (see figure 4), otherwise your scripts may not execute properly.

Most decent text editors (e.g. Notepad++, Sublime Text) have an option to set the line endings or convert to Unix. In the figure 5 it is shown the way to correct it in Notepad++.

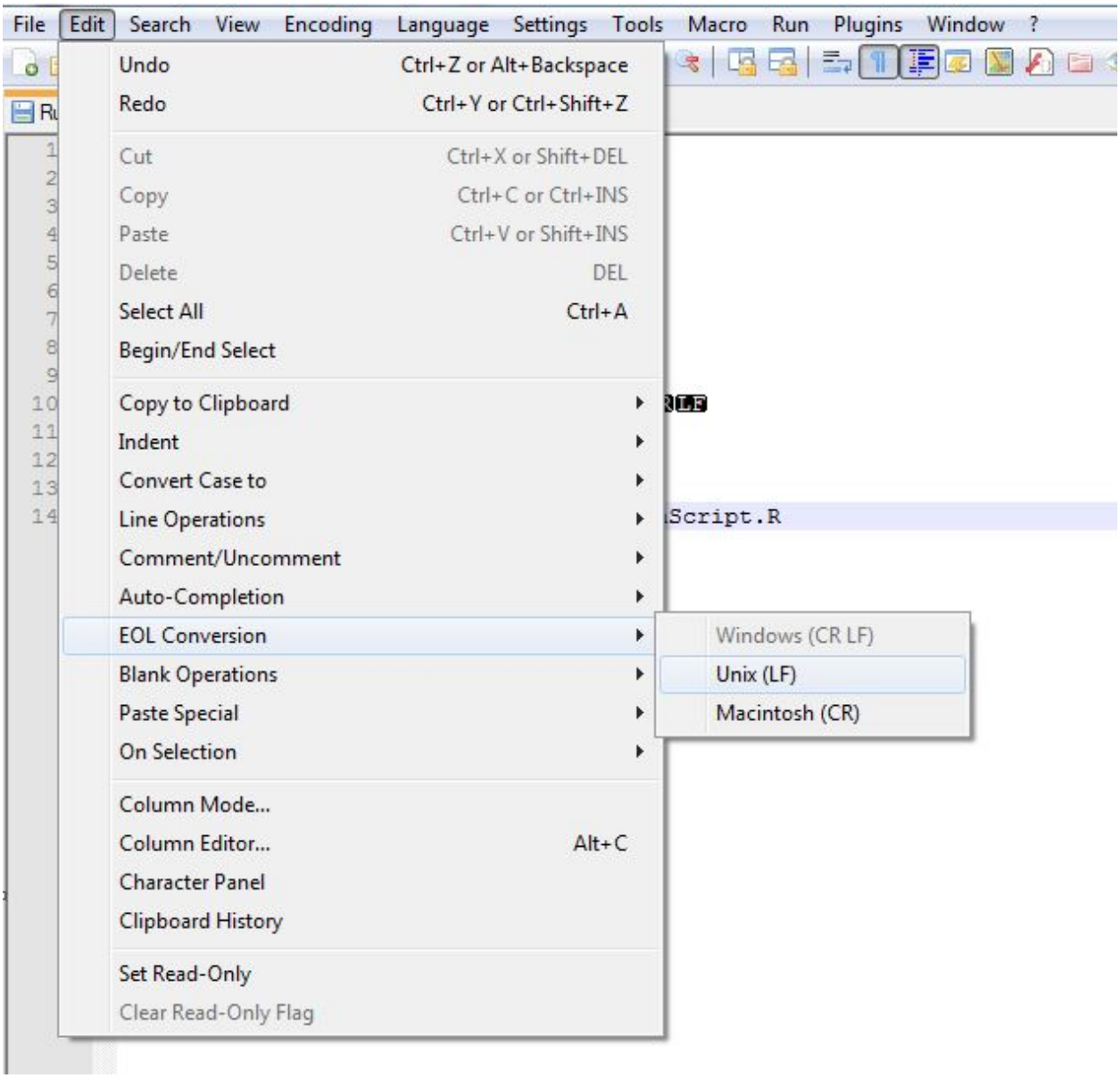

Figure 5: Modify Windows line endings with Notepad++

11 Useful Linux commands

pwd: It returns your working directory.
ls: It displays the content of your working directory.
cd: It changes your working directory to a new one given as parameter.
man: It opens the manual regarding the command given as parameter.
mkdir: It creates a new directory.
rmdir: It removes an empty directory. Use rm -r to delete the directory with the files in it.
nano: It opens nano, a text editor.
cp: It copies a directory. Add -r to copy the files in the folder.
exit: It exits from the terminal or ends the connection to SHARK.