INTRODUCTION TO RHADOOP

Janez Povh
EuroHPC Competence Center, February 3–4 2022

Outline

- Schedule
- Introduction to R
- Advanced and Big data management with R
- Big data management with RHadoop

Timetable

March 4th	
13:00–13:15	**Introduction to Day 2**
13:15–14:00	**Introduction to R**
14:00–14:15	*break*
14:15–15:00	**Advanced and Big data management with R** Dana manipulations with apply functions apply, lapply, sapply, vapply, tapply, and mapply. Big Data management with function for efficient parallel loops parLapply, parSapply, mcLapply and foreach-dopar.
15:00–15:15	*break*
15:15–16:00	**Big data management with RHadoop** Preparing and storing big data to HDFS using rhdfs library. Retriving from and managing big data in HDFS by plyrmr and rhdfs library.
16:00–16:15	*break*
16:15–17:00	**Big data analysis with RHadoop** Preparing map-reduce scripts to make basic data analysis tasks (extreme values, counts, mean values, dispersions, visualisations) using rhdfs library

Outline/next

- Schedule
- Introduction to R
- Advanced and Big data management with R
- Big data management with RHadoop

Introduction to R

What is R

- Software for Statistical Data Analysis
- Based on S
- Programming Environment
- Interpreted Language
- Data Storage, Analysis, Graphing
- Free and Open Source Software

How to obtain R

- R current version 4.1.3 (released on 2021-11-01).
- `http://cran.r-project.org`
- Binary source codes
- Windows executables

Pros and Cons

Pros:

- Free and Open Source
- Strong User Community
- Highly extensible, flexible
- Implementation of high-end statistical methods
- Flexible graphics and intelligent defaults

Cons

- Steep learning curve
- Slow for large datasets

Data types

- R Supports virtually any type of data
- Numbers, characters, logicals (TRUE/ FALSE)
- Arrays of virtually unlimited sizes
- Simplest: Vectors and Matrices
- Lists: Can Contain mixed type variables
- Data Frame: Rectangular Data Set

Data structures in R

Linear

- vectors (all same type)
- lists (mixed types)

Rectangular

- data frame
- matrix

Running R

- I recommend RStudio, an IDE for R.
- It is available as RStudio Desktop and **RStudio Server**, which runs on a remote server and allows accessing RStudio using a web browser.

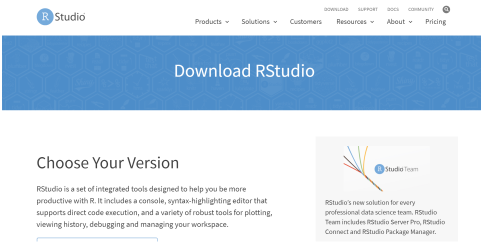


Figure 1: `https://rstudio.com/products/rstudio/download/`

RStudio on HPCFS

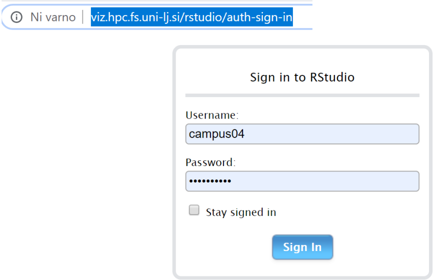
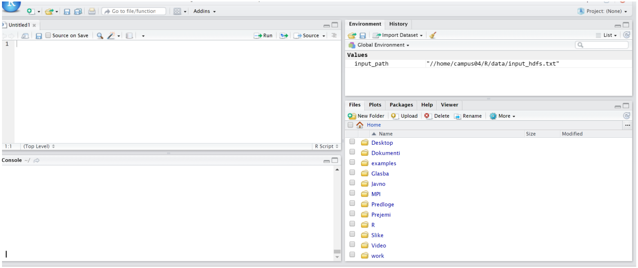


Figure 2: `http://viz.hpc.fs.uni-lj.si/rstudio/auth-sign-in`

RStudio on HPCFS



Before we start

Create directory for R scripts

```
work_dir=paste("/home", Sys.getenv("USER"),"resources", sep="/")
setwd(work_dir)
system("git pull")
```

The first R script file

- Open new script file CTRL+SHIFT+N
- Save the script file.

Create directory for R scripts

```
work_dir=paste("/home", Sys.getenv("USER"),"Rscripts/resources", sep="/")
work_dir=paste("/home", Sys.getenv("USER"),"Rscripts/big-data-training", sep="/")
data_dir=paste(work_dir,"data", sep="/")

#unlink(work_dir, recursive = TRUE)

ifelse(!dir.exists(work_dir), dir.create(work_dir), FALSE)
ifelse(!dir.exists(data_dir), dir.create(data_dir), FALSE)

setwd(work_dir)
system("git pull")
dir()
dir("data/")
```

Creating the first scrip file

Create and save simple data file

```
N=1000;
Data=data.frame(group=character(N),ints=numeric(N),reals=numeric(N))
Data$group=sample(c("a","b","c"), 1000, replace=TRUE);
Data$ints=rbinom(N,10,0.5);
Data$reals=rnorm(N);

head(Data)
Data

write.table(Data, file='Data/Data_Ex_1.txt', append = FALSE, dec = ".",col.names = TRUE)

ls()
rm(list = ls())
```

Load and analyse the data

Load data

```
Data_read<-read.table(file='data/Data_Ex_1.txt',header = TRUE)
# first few rows
head(Data_read)

#10 th row
Data_read[10,]
# column group
Data_read$group
Data_read[,1]
```

Load and analyse the data

Load data

```
  # compute means and counts by groups
  group count_ints mean_ints
a | 337 | 5.014837
b | 338 | 5.032544
c | 325 | 4.990769

# primitive solution
Group_lev=levels(Data_read$group)

Tab_summary=data.frame(group=character(3),count_ints=integer(3),mean_ints=numeric(3))
Tab_summary$group<-Group_lev
for (i in c(1:3)){
  sub_data = subset(Data_read,group==Group_lev[i])
  Tab_summary$count_ints[i]<-nrow(sub_data)
  Tab_summary$mean_ints[i]<-mean(sub_data$ints)
}
```

Analyse the data with dplyr, magrittr

- Library dplyr: ”select", "filter", "group_by","arrange", "mutate" and ”summarize".
- Library magrittr: "%>%"

dplyr

```
llibrary(dplyr)
library(magrittr)
Tab_summary1<-group_by(Data_read,group) %>% dplyr::summarise(count_ints=n(),mean_ints=
     mean(ints))

# other operations on rows and columns
Data_read_group_ints<-Data_read %>% select(group,ints)
# add new variable reals/ints
Data_read<-mutate(Data_read,ratio=reals/ints)
Data_read<-Data_read %>% mutate(ratio1=ints/reals)
#arrange
#sort accordind to increasing group
Data_read<-Data_read %>% arrange(desc(group))
Data_read<-Data_read %>% arrange(group)
```

Analyze the data by split, aggregate, sapply

split, aggregate, sapply

```
s <- split(Data_read, Data_read$group)
Tab_summary1<-t(sapply(s, function(x) return(c(mean(x$ints),length(x$group)) )))

Tab_summary2<-cbind(aggregate(ints~group,data = Data_read,FUN=length),aggregate(ints~
     group,data = Data_read,FUN=mean))
Tab_summary2<-Tab_summary2[,-3]
```

Outline/next

- Schedule
- Introduction to R
- Advanced and Big data management with R
- Big data management with RHadoop

Advanced and Big data management with R

 Advanced and Big data management with R

apply, lapply, sapply

apply, lapply, sapply

```
apply(X, MARGIN, FUN)
Here:
-x: an array or matrix
-MARGIN=1: the manipulation is performed on rows
-MARGIN=2: the manipulation is performed on columns
-MARGIN=c(1,2): the manipulation is performed on rows and columns
-FUN: tells which function to apply. Built functions like mean, median, sum, min, max and
      even
user-defined functions can be applied
```

apply

For data constructed above (Data_read) compute row and columns means using apply

apply

```
Data_read<-read.table(file='data/Data_Ex_1.txt',header = TRUE)

Data_col_means_1 <- colMeans(Data_read[,-1])
Data_col_means_2 <- apply(Data_read[,-1],2,FUN =mean)

Data_row_means_1 <- rowMeans(Data_read[,-1])
Data_row_means_2 <- apply(Data_read[,-1],1,FUN =mean)

Data_both_squares <- apply(Data_read[,-1],c(1,2),FUN = function(x) return(x^2))
```

 Advanced and Big data management with R

lapply

- `lapply` function takes list, vector or data frame as input and returns only list as output
- `sapply` function takes list, vector or data frame as input. It is similar to lapply function but returns only vector as output.

For data constructed above (`Data_read`) compute row and columns sums using `lapply`

lapply

```
Data_col_sums_1 <- apply(Data_read[,-1],2,FUN =sum)
Data_col_sums_2 <- lapply(Data_read[,-1],FUN =sum)

typeof(Data_col_sums_1)
typeof(Data_col_sums_2)


Data_abs <- lapply(Data_read[,-1],FUN =abs)
Data_sq <- lapply(Data_read[,-1],FUN = function(x){x^2})

typeof(Data_abs)
length(Data_abs)
```

 Advanced and Big data management with R

sapply

For data constructed above (Data_read) compute row and columns sums using sapply

sapply

```
Data_col_sums_1 <- apply(Data_read[,-1],2,FUN =sum)
Data_col_sums_2 <- lapply(Data_read[,-1],FUN =sum)
Data_col_sums_3 <- sapply(Data_read[,-1],FUN =sum)

typeof(Data_col_sums_1)
typeof(Data_col_sums_2)
typeof(Data_col_sums_3)


Data_col_sums_4 <- lapply(list(Data_read$ints,Data_read$reals),FUN =sum)
Data_col_sums_5 <- sapply(list(Data_read$ints,Data_read$reals),FUN =sum)
Data_col_len_1 <- lapply(list(Data_read$ints,Data_read$reals),FUN =length)
Data_col_len_2 <- sapply(list(Data_read$ints,Data_read$reals),FUN =length)
```

 Advanced and Big data management with R

for loop

Let us compute sums of all elements of 12 random matrices of order 3000×3000

for

```
N=3000
set.seed(2021)
sum_rand=rep(0,11);
tic()
for (i in c(1:12)){
  A=randn(N,N)
  sum_rand[i]=sum(A)
}
time_for=toc()
```

 Advanced and Big data management with R

foreach do loop

Let us compute sums of all elements of 12 random matrices of order 3000×3000

for

```
N=3000
set.seed(2021)
sum_rand=rep(0,11);
tic()
foreach (i = c(1:12)) %do% {
  A=randn(N,N)
  sum_rand[i]=sum(A)
}
time_foreach=toc()
```

Parallel foreach dopar loop

Let us compute sums of all elements of 12 random matrices of order 3000×3000 using `foreach ...dopar` from `foreach` and `doParallel`

for

```
N=3000
set.seed(2021)
sum_rand=rep(0,11);
tic()
foreach (i = c(1:12)) %dopar% {
  A=randn(N,N)
  sum_rand[i]=sum(A)
}
time_foreach_dopar=toc()
```

Do you observe any difference?

Parallel foreach dopar loop

Let us compute sums of all elements of 12 random matrices of order 3000×3000 using `foreach ...dopar` from `foreach`, `doParallel`. Create cluster!

for

```
N=3000
set.seed(2021)
registerDoParallel(12)  # use multicore, set to the number of our cores - needed for
      foerach dopar

sum_rand=rep(0,11);
tic()
foreach (i = c(1:12)) %dopar% {
   A=randn(N,N)
   sum_rand[i]=sum(A)
}
time_foreach_dopar_1=toc()
registerDoSEQ()
```

Do you observe any difference?

 Advanced and Big data management with R

Library parallel

- encapsulates existing libraries `multicore, snow`
- two ways of parallelization:
  - The socket approach: launches a new version of R on each core via networking (e.g. the same as if you connected to a remote server), but the connection is happening all on your own computer.
    - pros: (i) Works on any system (including Windows); (ii) Each process on each node is unique so it can't cross-contaminate.
    - cons: (i) Each process is unique so it will be slower (ii) Things such as package loading need to be done in each process separately. Variables defined on your main version of R don't exist on each core unless explicitly placed there. (iii) More complicated to implement.
  - use `parLapply, parSapply`

 Advanced and Big data management with R

Library `parallel`

- The forking approach copies the entire current version of R and moves it to a new core.
  - (i) Faster than sockets. (ii) Because it copies the existing version of R, your entire workspace exists in each process. (iii) Easy to implement.
  - Cons (i) Only works on POSIX systems (Mac, Linux, Unix, BSD) and not Windows. (ii) it can cause issues specifically with random number generation or when running in a GUI (such as RStudio). This doesn't come up often.
- use `mclapply`

Parallel versions of lapply

By using library parallel and parSapply, mclapply compute sums of all elements of 12 random matrices of order 3000×3000. Create cluster!

parallel versions of apply

```
mat_sum<-function(x){
  A=rand(x)
  return(sum(A))
}
tic()
time_lapply<-system.time({
  set.seed(2021)
  sum_rand_lapply=lapply(rep(3000,12),FUN=mat_sum)
  time_lapply=toc()
})

time_sapply<-system.time({
  set.seed(2021)
  sum_rand_sapply=sapply(rep(3000,12),FUN=mat_sum)
})
```

Parallel versions of lapply

parallel versions of apply

```
time_mcLapply<-system.time({
  set.seed(2021)
  sum_rand_mcLapply=mclapply(X=rep(3000,12),FUN=mat_sum,mc.cores = 12)
})

time_parLapply<-system.time({
  clust <- makeCluster(12, type="PSOCK")
  set.seed(2021)
  sum_rand_parLapply=parLapply(cl,rep(3000,1000),fun=mat_sum)
  stopCluster(clust)
})

time_parSapply<-system.time({
  clust <- makeCluster(12, type="PSOCK")
  set.seed(2021)
  sum_rand_parSapply=parSapply(cl,rep(3000,20),FUN=mat_sum)
  stopCluster(clust)
})
```

 Advanced and Big data management with R

Parallel versions of lapply

parallel versions of apply

```
times_apply<-rbind(time_lapply,time_sapply,time_parLapply,time_parSapply,time_mcLapply)

> times_apply[,1:3]
               user.self sys.self elapsed
time_lapply        5.120    0.954   6.072
time_sapply        5.049    0.885   5.932
time_parLapply     0.076    0.209  47.999
time_parSapply     0.021    0.105   4.286
time_mcLapply      0.003    0.040   0.531
```

Libraries for shared memory parallelization in R

- Parallel for-loop (`foreach...dopar`). Cluster created by `registerDoParallel(N)` and `registerDoSEQ()`. Library `foreach`, `doParalel` needed.
- Parallel `apply`: `parLapply`, `parSapply`, `mcLapply` need library parallel.

Outline/next

- Schedule
- Introduction to R
- Advanced and Big data management with R
- Big data management with RHadoop

Big data management with RHadoop

The goals of the second part

- Demonstrating basic data management operations with RHadoop;
- By few examples showing basic data analysis with RHadoop;

Motivation

- Do data analysis (statistics), do not bother with low level settings
- Stay within `R` (and `RStudio`)

Overall picture

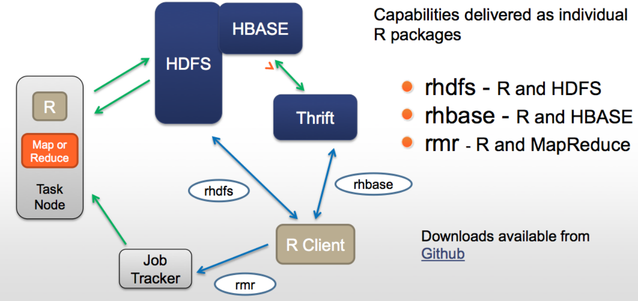
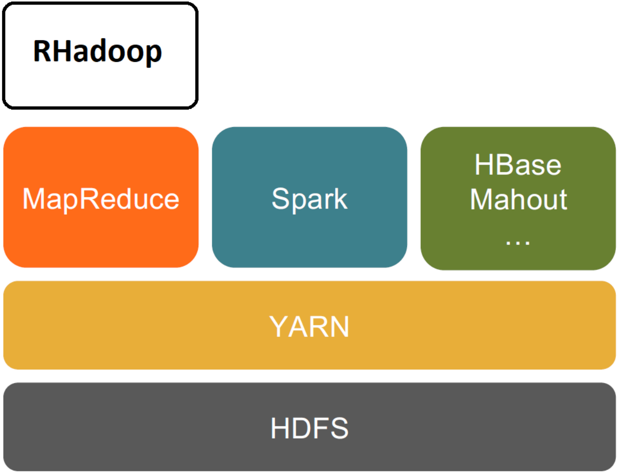


Figure 3:
https://www.r-bloggers.com/slides-and-replay-from-r-and-hadoop-webinar/

Overall picture



 Big data management with RHadoop

First little example

content...

 Big data management with RHadoop

Setting up RHadoop using terminal window

```
export LD_LIBRARY_PATH=/opt/apps/software/Java/1.7.0_80/lib:${LD_LIBRARY_PATH}
export PATH=/opt/apps/software/Java/1.7.0_80:${PATH}
export JAVA_HOME=/opt/apps/software/Java/1.7.0_80
export PATH=/opt/apps/software/Hadoop/2.6.0-cdh5.8.0-native/bin:${PATH}
export PATH=/opt/apps/software/Hadoop/2.6.0-cdh5.8.0-native/sbin:${PATH}
export LD_LIBRARY_PATH=/opt/apps/software/Hadoop/2.6.0-cdh5.8.0-native/lib:${LD_LIBRARY_PATH}
export HADOOP_HOME=/opt/apps/software/Hadoop/2.6.0-cdh5.8.0-native/share/hadoop/mapreduce
```

Rhadoop

5 R packages provided by RevolutionAnalytics[1][2]:

- rhdfs - basic connectivity to the Hadoop Distributed File System (browse, read, write, and modify files stored in HDFS)
- rhbase - basic connectivity to the HBASE distributed database, using the Thrift server.
- plyrmr - enables the R user to perform common data manipulation operations, as found in plyr and reshape2
- rmr2 - allows R developer to perform statistical analysis in R via Hadoop MapReduce functionality on a Hadoop cluster.
- ravro - adds the ability to read, write and manipulate avro files from local and HDFS file system.

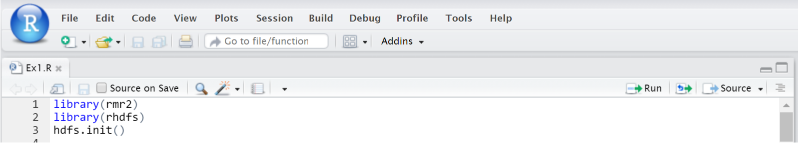
[1] https://github.com/RevolutionAnalytics

[2] https://github.com/RevolutionAnalytics/RHadoop/wiki/Downloads

 Big data management with RHadoop

Setting up the Rhadoop - cnt.

- Establish the connectivity to the Hadoop Distributed File System by loading the library rhdfs. `library(rhdfs)`
- Load libraries to work with Hadoop MapReduce `library(rmr2)`
- Initialize HDSF `hdfs.init()`.
- All together:



Basic data operations with RHadoop.

EURO

List files in the root directory of DFS `hdfs.ls("/")`

```
> hdfs.ls("/")
permission  owner      group      size           modtime                      file
1 -rw-r--r-- hadoop supergroup 184814018 2021-09-25 22:16       /BigData_reg_class
2 -rw-r--r-- hadoop supergroup  33602002 2021-09-25 22:16               /CEnetBig
3 -rw-r--r-- hadoop supergroup 476054348 2021-09-25 22:16 /electricity-energy.txt
4 drwxrwxrwx hadoop supergroup         0 2021-09-28 02:14                    /tmp
5 drwxr-xr-x hadoop supergroup         0 2021-09-25 11:49                   /user
```

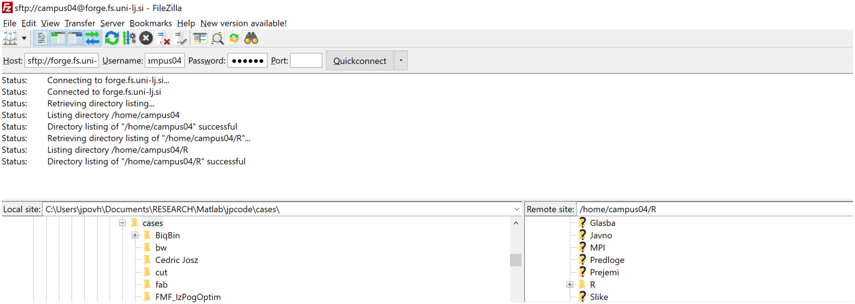
Basic data operations with RHadoop.

List files in the home directory of each user
`hdfs.ls("/user/campus01")`

```
hdfs.ls("/user/campus01")
permission    owner   group         size          modtime
     file
1 -rw-r--r-- campus01 hadoop       12466 2020-09-16 06:47  /user/campus01/
     OurSmallData
2 -rw-r--r-- campus01 hadoop 18836041094 2020-09-11 09:16  /user/campus01/safecast.
     csv
3 -rw-r--r-- campus01 hadoop   336031560 2020-09-15 15:30     /user/campus01/
     wiki321MB
4 drwxr-xr-x campus01 hadoop           0 2020-09-15 15:30 /user/campus01/wordcount_
     out
```

 Big data management with RHadoop

Moving data around - FileZilla



Big data management with RHadoop

Moving data around with Linux

Copy from other account

```
cp /home/campus01/R/data/iris.csv /home/campusxx/R/data/iris.csv
```

Copy from internet

```
curl -o  /home/campus01/R/data/iris.csv
 https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/
 639388c2cbc2120a14dcf466e85730eb8be498bb/iris.csv
```

 Big data management with RHadoop

Moving data around with Linux or RHadoop

Copy from internet address or local folder to hdfs within RHadoop

```
curl https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/639388
     c2cbc2120a14dcf466e85730eb8be498bb/iris.csv |
hadoop fs -appendToFile - /user/campus01/iris.csv

system('curl https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/639388
     c2cbc2120a14dcf466e85730eb8be498bb/iris.csv |
hadoop fs -appendToFile - /user/campus01/iris.csv')

system('curl file:///home/campus01/R/data/iris.csv | hadoop fs -appendToFile - /user/
     campus01/iris.csv')

system('hadoop fs -appendToFile  /home/campus01/R/data/iris.csv /user/campus01/iris.csv')
```

 Big data management with RHadoop

Create and store data in HDFS

Use small data created at the beginning and stored as

```
file_name = paste("/home", Sys.getenv("USER"),'myRscripts','Data_Ex_1.txt', sep="/")
Data_read<-read.table(file=file_name,header = TRUE)

myDFS_File=paste("/user", Sys.getenv("USER"), "OurSmallData", sep="/")
hdfs.rm(myDFS_File)
OurSmallData=to.dfs(Data_read, myDFS_File,format="native")
SmallData1_DFS=from.dfs(OurSmallData)
system("hdfs fsck /user/campus01/OurSmallData")
```

CEnetBig

`CEnetBig` contains data about customers of company X: for each customer we have one row containing

- ID of the customer;
- the values of their bills for period January 2016-December 2016;
- type of product that they have;

```
       id 2016_1 2016_2 2016_3 2016_4 2016_5 2016_6 2016_7 2016_8 2016_9 2016_10 2016_
    11 2016_12 type
[1,] 100373 137.66 141.57 128.83 133.00  97.39 116.62 123.97 156.83  90.50   98.62
    118.61  152.34    4
[2,] 100194  98.32 119.40 120.30 105.67  90.26  80.13  80.62 108.63 104.30  123.31
    101.93  140.85    2
[3,] 100565 127.60 133.79  90.15  62.33  87.96  92.20  72.04 113.69  65.95   82.69
    85.72  121.81    2
```

HDFS statistics for CEnetBig

- From RStudio `system("hdfs fsck /CEnetBig")`
- From command line: `hadoop fsck /CEnetBig`

```
> system("hdfs fsck /CEnetBig")
Connecting to namenode via http://viz.hpc:50070
FSCK started by campus01 (auth:SIMPLE) from /10.0.2.99 for path /CEnetBig at Tue
    Sep 28 07:45:39 CEST 2021
.Status: HEALTHY
Total size: 33602002 B
Total dirs: 0
Total files:  1
Total symlinks:   0
Total blocks (validated): 1 (avg. block size 33602002 B)
Minimally replicated blocks:  1 (100.0 %)
Over-replicated blocks: 0 (0.0 %)
Under-replicated blocks:  0 (0.0 %)
Mis-replicated blocks:    0 (0.0 %)
Default replication factor: 2
Average block replication:  2.0
Corrupt blocks:   0
Missing replicas:   0 (0.0 %)
Number of data-nodes:   20
Number of racks:    1
FSCK ended at Tue Sep 28 07:45:39 CEST 2021 in 2 milliseconds

The filesystem under path '/CEnetBig' is HEALTHY
```

HDFS statistics for CEnetBig

EURO

- From RStudio `system("hdfs fsck /user/jpovh/safecast.csv")`

```
Connecting to namenode via http://viz.hpc:50070
FSCK started by campus01 (auth:SIMPLE) from /10.0.2.99 for path /user/campus01/
     safecast.csv at Wed Sep 16 07:39:21 CEST 2020
.Status: HEALTHY
Total size: 18836041094 B
Total dirs: 0
Total files:  1
Total symlinks:   0
Total blocks (validated): 141 (avg. block size 133588943 B)
Minimally replicated blocks:  141 (100.0 %)
Over-replicated blocks: 0 (0.0 %)
Under-replicated blocks:  0 (0.0 %)
Mis-replicated blocks:    0 (0.0 %)
Default replication factor: 3
Average block replication:  3.0
Corrupt blocks:   0
Missing replicas:   0 (0.0 %)
Number of data-nodes:   16
Number of racks:    8
FSCK ended at Wed Sep 16 07:39:21 CEST 2020 in 10 milliseconds

The filesystem under path '/user/campus01/safecast.csv' is HEALTHY
```

CEnetBig

- Load data into active memory:

```
CEnetBig<-from.dfs("/CEnetBig")
```

- `CEnetBig` is a key-value pair with void key.

```
> CEnetBig$key
NULL
> CEnetBig$val[1:3,]
id 2016_1 2016_2 2016_3 2016_4 2016_5 2016_6 2016_7 2016_8 2016_9 2016_10 2016_11
     2016_12 type
[1,] 100373 137.66 141.57 128.83 133.00  97.39 116.62 123.97 156.83  90.50   98.62
      118.61  152.34    4
[2,] 100194  98.32 119.40 120.30 105.67  90.26  80.13  80.62 108.63 104.30  123.31
      101.93  140.85    2
[3,] 100565 127.60 133.79  90.15  62.33  87.96  92.20  72.04 113.69  65.95   82.69
       85.72  121.81    2
```

First Big Data challenge

Goal: In the column 2016_1 find the maximum value.

Use: $\max\{\cup_i A_i\} = \max_i\{\max A_i\}$.

$$\begin{aligned} 9 &= \max\{1,5,4,7,9,2,3,5\} \\ &= \underbrace{\max\{1,5,4,7\},\max\{9,2,3,5\}}_{\max} \end{aligned}$$

Suppose XX is submatrix of `CEnetBig` of 1st 100 rows. We find the maximum of column 2016_1 by

```
XX=CEnetBig$val[1:100,]
M=max(XX[,"2016_1"])
```

 Big data management with RHadoop

Finding maximum by Map-Reduce

EURO

- MAP:

```
mapper = function (., X) {
  M=max(X[,"2016_1"]);
  keyval(1,M)
}
```

- REDUCE:

```
reducer = function(k, A) {
  keyval(k, list(Reduce("max", A))) # take maximum of maxima
}
```

 Big data management with RHadoop

Finding maximum by Map-Reduce - cnt.

EURO

- MAP-REDUCE:

```
GlobalMaxMR = from.dfs(
  mapreduce(
  input = "/CEnetBig",
  map = mapper,
  reduce = reducer
  )
)
```

- Final code:

```
GlobMax =GlobalMaxMR$val
```

- Result

```
> GlobalMaxMR$val
[[1]]
[1] 243.25
```

Finding maximum, number of map calls and block sizes

EURO

```
mapper2 = function (., X) {
  M=max(X[,"2016_1"]);
  keyval(1:3,list(1,M,dim(X)[1]))
}

reducer2 = function(k, A) {
  if(k==1){
    keyval(k, list(Reduce("+", A))) # take sum
  } else if (k==2) {
    keyval(k, list(Reduce("max", A))) # take maximum of maxima
  } else {
    keyval(k, A)
  }
}

GlobalMaxNumMR = from.dfs(
  mapreduce(
    input = "/CEnetBig",
    map = mapper2,
    reduce = reducer2
  )
)
```

 Big data management with RHadoop

Finding maximum, number of map calls and block sizes - cnt.

```
> GlobalMaxNumMR
$key
[1]  1  2  3  3  3  3  3  3  3  3  3  3  3  3

$val
$val[[1]]
[1]  12

$val[[2]]
[1]  243.25

$val[[3]]
[1]  89285

$val[[4]]
[1]  89284

$val[[5]]
[1]  80356

...

$val[[10]]
[1]  89285

$val[[11]]
[1]  89285

$val[[12]]
[1]  89285
```

Second Big Data challenge

Goal: Compute the mean value of the column 2016_1 ..

Note: $\bar{x} = \sum_i X_i/n$

Suppose XX is submatrix of CEnetBig of 1st 100 rows. We find mean value of column 2016_1 by

```
XX=CEnetBig$val[1:100,]
m=mean(XX[,"2016_1"])
```

- If s_i and n_i are sums and sizes of blocks of data, respectively, then the mean value of all data is

$$\bar{x} = \frac{\sum_i s_i}{\sum_i n_i}$$

Finding mean value by Map-Reduce

- MAP:

```
mapper_mean = function (., X) {
  n=nrow(X);
  mi=sum(X[,2]);
  keyval(1:2,list(n,mi));
}
```

- REDUCE:

```
reducer_mean = function(k, A) {
  keyval(k,list(Reduce('+', A)))
}
```

Finding mean value by Map-Reduce - cnt.

- MAP-REDUCE:

```
Block_means <-   from.dfs(
  mapreduce(
    input = "/CEnetBig",
    map = mapper_mean,
    reduce = reducer_mean
  )
)
```

- Final code:

```
GlobalMean=Block_means$val[[2]]/Block_means$val[[1]]
```

- Result

```
> GlobalMean
[1] 129.4716
```

 Big data management with RHadoop

Third Big Data challenge

EURO

Goal: Compute the variance of σ^2 of the CEnetBig[,3] .

Note: $\sigma^2 = \frac{\sum_k (X_{k,2} - \bar{x}_2)^2}{n} = \frac{\sum_k X_{k,2}^2}{n} - \bar{x}_2^2$.

 Big data management with RHadoop

Third Big Data challenge - cnt.

```
mapper_var = function (., X) {
  n=nrow(X);
  mi=sum(X[,2]);
  si=sum(X[,2]^2);
  keyval(1:3,list(n,mi,si));
}


reducer_var = function(k, A) {
  keyval(k,list(Reduce('+', A)))
}

Block_var <-   from.dfs(
  mapreduce(
    input = "/CEnetBig",
    map = mapper_var,
    reduce = reducer_var
  )
)

globalVar=Block_var$val[[3]]/Block_var$val[[1]]-(Block_var$val[[2]]/Block_var$val[[1]])^2
> globalVar
[1] 595.1341
```

 Big data management with RHadoop

Fourth Big Data challenge

EURO

Goal: Compute the covariance matrix Σ of the CEnetBig[,2:13] .

Note: $\Sigma_{ij} = \frac{\sum_k (X_{ik} - \bar{x}_i)(X_{jk} - \bar{x}_j)}{n} = \frac{1}{n}(\tilde{X}^T \tilde{X})_{ij}$.

Suppose XX is submatrix of `CEnetBig` of 1st 100 rows and with columns `'2016_1',...,'2016_12'`. We find covariance matrix of XX

```
XX=CEnetBig$val[1:100,2:13]
Sigma=cov(XX)
```

Note: Naive approach will visit the data several times.

Third Big Data challenge - cnt.

EURO

```
> Sigma
2016_1    2016_2    2016_3    2016_4    2016_5    2016_6    2016_7     2016_8    2016_9   2016_10
      2016_11   2016_12
2016_1   554.66627 197.7795 144.7789 131.1854 249.1535 124.1262 252.6528  53.31369
     199.2839 120.2593 257.9729 158.0299
2016_2   197.77949 687.8934 302.7297 307.0862 266.9029 261.8073 280.3199 252.36691
     274.6391 247.4709 310.5588 140.8925
2016_3   144.77895 302.7297 762.0102 284.1748 247.8277 175.4163 283.0150 217.00145
     321.8898 244.9201 413.3578 173.4369
2016_4   131.18542 307.0862 284.1748 605.7750 169.2399 253.4410 292.7296 209.68617
     283.8475 247.4226 422.2579 219.1580
2016_5   249.15355 266.9029 247.8277 169.2399 541.3642 171.9361 227.3288 194.71391
     293.5147 218.3279 253.6789 219.2686
2016_6   124.12617 261.8073 175.4163 253.4410 171.9361 567.5522 232.6065 183.04757
     219.4846 192.3792 272.8218 140.0295
2016_7   252.65276 280.3199 283.0150 292.7296 227.3288 232.6065 681.2422 261.19614
     293.7390 211.6760 450.0655 208.6689
2016_8    53.31369 252.3669 217.0015 209.6862 194.7139 183.0476 261.1961 639.62214
     260.6902 101.4208 189.6450 187.1990
2016_9   199.28392 274.6391 321.8898 283.8475 293.5147 219.4846 293.7390 260.69023
     635.4909 186.6704 370.9400 294.8569
2016_10 120.25931 247.4709 244.9201 247.4226 218.3279 192.3792 211.6760 101.42076
     186.6704 706.0847 296.6746 169.5678
2016_11 257.97290 310.5588 413.3578 422.2579 253.6789 272.8218 450.0655 189.64504
     370.9400 296.6746 877.7393 243.8821
2016_12 158.02993 140.8925 173.4369 219.1580 219.2686 140.0295 208.6689 187.19898
     294.8569 169.5678 243.8821 561.2406
```

 Big data management with RHadoop

Covariance matrix - cnt.

- Some mathematics:

$$\Sigma_{ij} = \frac{\sum_k (X_{ik} - \bar{x}_i)(X_{jk} - \bar{x}_j)}{n} = \frac{\sum_k X_{ik} X_{jk}}{n} - \bar{x}_i \bar{x}_j .$$

$$\Sigma = \frac{1}{n} X^T X - \bar{x}\bar{x}^T$$

- Block structure: Suppose we decompose

$$X = \begin{bmatrix} X^1 \\ X^2 \\ \vdots \\ X^k \end{bmatrix}$$

where X^i is a block of X having n_i rows.

- The "tough" product rewrites as

$$X^T X = \sum_{i=1}^{k} (X^i)^T X^i .$$

Covariance matrix - cnt.

- Similarly: if n_i, s_i are row-sizes and column sums of blocks X^i

$$\bar{x} = \frac{\sum_i s_i}{\sum_i n_i}. \tag{1}$$

```
mapperSS = function (., X) {
  ni=nrow(X);
  si=colSums(X[,2:13]);
  SSi=t(X[,2:13])%*%X[,2:13];
  keyval(1:3,list(ni,si,SSi));
}
```

- REDUCE:

```
reducerSS = function(k, A) {
  keyval(k,list(Reduce('+', A)))
}
```

Covariance matrix - cnt.

- MAP-REDUCE:

```
CovMatrixRaw <-   from.dfs(
  mapreduce(
    input = "/CEnetBig",
    map = mapperSS,
    reduce = reducerSS
  )
)
```

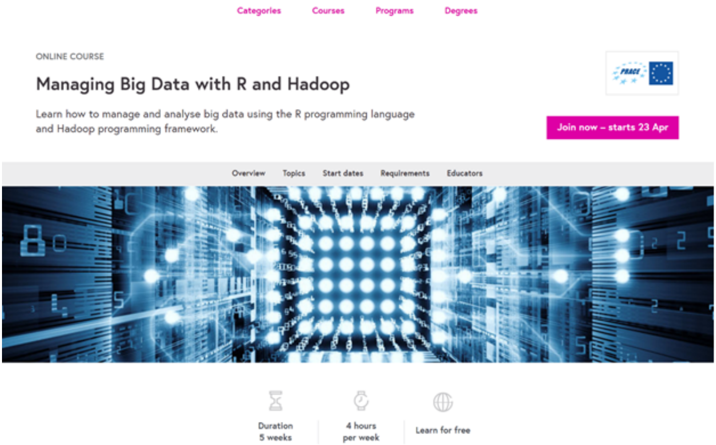
- Final code

```
meanVec <- CovMatrixRaw$val[[2]]/CovMatrixRaw$val[[1]]
CovMat <- CovMatrixRaw$val[[3]]/CovMatrixRaw$val[[1]] -outer(meanVec,meanVec)
```

 Big data management with RHadoop

MOOC

Visit our MOOC:



Challenge

Count the number of consumers with total consumption larger than 1500.

 Big data management with RHadoop

Challenge

For each type find a list of consumers having this type of contract.

 Big data management with RHadoop

Word count example

Count the words in text document by Map-Reduce

Word count

```
library(readr)
library(rmr2)
library(rhdfs)
hdfs.init()
#rmr.options(backend = "local")
rmr.options(backend = "hadoop")
ebookLocation_hdfs <- "/public/ullyses.txt"
wikiLocation_hdfs <- "/public/wiki_1k_lines"
m <- mapreduce(input = ebookLocation_hdfs,
#                output = ebookLocation_hdfs,
               input.format  =  "text",
               map = function(k, v){
                 words <- unlist(strsplit(v, split = "[[:space:][:punct:]]"))
                 words <- tolower(words)
                 words <- gsub("[0-9]", "", words)
                 words <- words[words != ""]
                 wordcount <- table(words)
                 keyval(
                   key = names(wordcount),
                   val = as.numeric(wordcount)
                 )
               },
               reduce = function(k, counts){
                 keyval(key = k
```

 Big data management with RHadoop

Word count example

Count the words in text document by Map-Reduce

Word count

```
# Retrieve results and prepare to plot ------------------------------------
x <- from.dfs(m)
dat <- data.frame(
  word  = keys(x),
  count = values(x)
)
dat <- dat[order(dat$count, decreasing=TRUE), ]
> head(dat, 6)
    word count
825  the 15130
121   of  8260
201  and  7285
1      a  6581
152   to  5043
93    in  5004
```