1 Topic

MapReduce with R using the « rmr2 » package.

Big Data[1] is a very popular topic these last years[2]. The **big data analytics** refers to the process to discovering useful information or knowledge from big data. That is an important issue for organizations[3]. In concrete terms, the aim is to extend, adapt or even create novel exploratory data analysis or data mining approaches to new data sources of which the main characteristics are “volume”, “variety” and “velocity”.

Distributed computing is essential in the big data context. It is illusory to want infinitely increase the power of servers for following the exponential growth of information to process. The solution depends on the efficient cooperation of a myriad of networked computers, ensuring both the volume management and computing power. **Hadoop** is a solution commonly cited for this requirement. This is a set of algorithms (an open-source software framework written in Java) for distributed storage and distributed processing of very large data sets (Big Data) on computer clusters built from commodity hardware[4]. For the implementation of distributed programs, the **MapReduce** programming model plays an important role[5]. The processing of large dataset can be implemented with parallel algorithms on a cluster of connected computers (nodes).

In this tutorial, we are interested in MapReduce programming in R. We use the technology RHadoop[6] of the Revolution Analytics Company. The "**rmr2**" package in particular allows to learn the MapReduce programming without having to install the Hadoop environment which is already sufficiently complicated. There are some tutorials about this subject on the web. The one of Hugh Devlin (January 2014) is undoubtedly one of the most interesting[7]. But, it is perhaps more sophisticated for the students which are not very familiar with the programming in R. So I

[1] http://en.wikipedia.org/wiki/Big_data

[2] http://www.google.fr/trends/explore#q=big%20data

[3] http://www.sas.com/en_us/insights/analytics/big-data-analytics.html

[4] http://en.wikipedia.org/wiki/Apache_Hadoop

[5] http://en.wikipedia.org/wiki/MapReduce

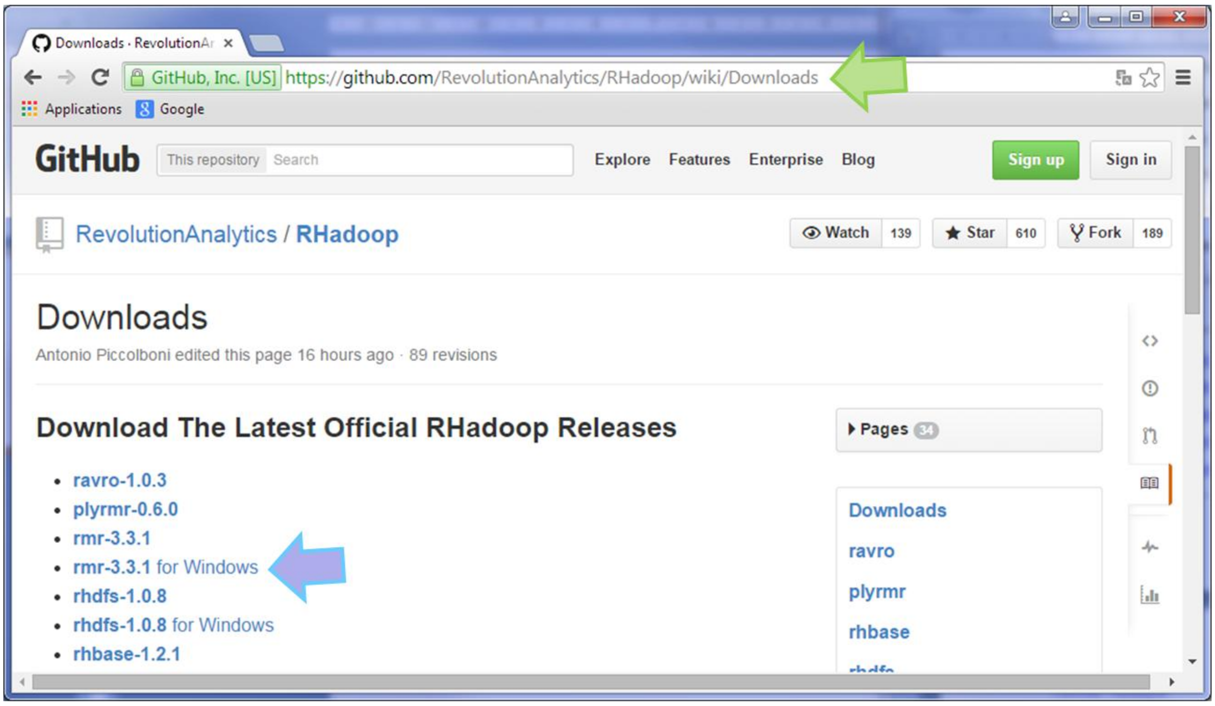
[6] http://blog.revolutionanalytics.com/2011/09/mapreduce-hadoop-r.html

[7] https://github.com/RevolutionAnalytics/rmr2/blob/master/docs/tutorial.md

decided to start afresh with very simple examples in a first time. Then, in a second time, we progress by programming a simple data mining algorithm such as the multiple linear regression.

2 Installing the package « rmr2 »

A tutorial described the steps for Windows[8]. It seems that the package works only under 64 bit system. I have installed first the package "functional". Then, I download the "**rmr2**" file from GitHub (**rmr2_3.3.1.zip**). I installed manually the package by using the command "Install package from ZIP file" of R. I use **64 bit - Windows 7** and **R 3.1.2**.



We then execute the following commands to prepare the ground.

```
#load the package rmr2
library(rmr2)

#in order to work without the Hadoop environment
rmr.options(backend="local")
```

The second command gives us the opportunity to practice the MapReduce programming without having to install the Hadoop environment.

8 http://tuxette.nathalievilla.org/?p=1455&lang=en#win

3 MapReduce – Processing a vector

MapReduce libraries perform a lot of tasks to insure the security and the performance of the system. But, in the programmer point of view, it is fairly straightforward. The MapReduce programming consists in decomposing a task in two sub-tasks: map and reduce. In the **MAP** step, the node analyzes the problem. It slices it into subproblems and delegates them to other nodes (which can also make the same recursively). Each subproblem is associated to a key that allows to identify the part of the treated problem. These subproblems are processed with the **REDUCE** procedure. It returns the result that we can identify with a key. The **<key, value>** pairs play a very important role in this framework. "Value" can refer to data or to results.

3.1 First program – Counting values

In this section, we handle a vector of integer numbers. We want to distinguish between odd and even values, send them on 2 different nodes, and perform processing on each of the sub sets.

```
#vector of integer values
x <- c(2,6,67,85,7,9,4,21,78,45)
```

We have 4 even values and 6 odd values.

MAP. We written the following **map()** function. It uses the modulo (%%) operator to check if there is a remainder when dividing by 2. If the remainder is zero, the number is even, the value 1 is generated as key. The used key is 2 when the number is odd.

```
#map function
map_valeurs <- function(., v){
  #calculate the key
  cle <- ifelse (v %% 2 == 0, 1, 2)
  #return the key and the value
  return(keyval(cle,v))
}
```

The MAP function usually takes two inputs: the key and the value to process. In our case, we use the value to generate the key. The first parameter is therefore ignored.

The input "v" in our case represents a vector to analyze. The variable "cle" (key) generated by ifelse() is a vector.

The function **keyval**() returns the key-value pairs i.e. it associates a key to each value of the input vector v. We obtain the following pair of vectors for our dataset.

Cle (key)	v
1	2
1	6
2	67
2	85
2	7
2	9
1	4
2	21
1	78
2	45

MAPREDUCE uses these two vectors in order to partition the initial vector in two subsets of numbers: (2, 6, 4, 78) for the even value, with the key = 1; (67, 85, 7, 9, 21, 45) for the odd values, with the key = 2.

REDUCE. In the reduce step, the subset are processed on the nodes in the cluster. The reduce() function is therefore called as many times as there are different values of key.

```
#reduce function
reduce_valeurs <- function(k, v){
  #length of the vector
  nb <- length(v)
  #return the key and the corresponding value (result)
  return(keyval(k, nb))
}
```

The key is needed to know what subset is processed. We calculate its length with **length**(). We send the result with **return**() by combining the key and the result of the calculations with the **keyval**() function.

MAPREDUCE. Now let us see how to organize all of this using the rmr2 mapreduce() function.

```
#transform the data in a type recognized by rmr2
x.dfs <- to.dfs(x)
```

```
#call the mapreduce function of rmr2
calcul <- mapreduce(input = x.dfs, map = map_valeurs, reduce = reduce_valeurs)

#transform the result in a type recognized by R
resultat <- from.dfs(calcul)

#printing
print(resultat)

#class of the result
print(class(resultat))
```

to.dfs() transforms the dataset in a format recognized by rmr2; **from.dfs()** performs the inverse operation i.e. it transforms the rmr2 object in a format recognized by R (a list as we see below).

The **mapreduce()** function is essential. It takes three parameters here:

- "input" is the dataset to process;
- "map" is the function called to map the data in key-value pairs;
- "reduce" processes the subset of data and returns the result with its key.

At the end of processing, we obtain under R:

```
> #affichage
> print(resultat)
$key
[1] 1 2

$val
[1] 4 6

>
> #classe de resultat
> print(class(resultat))
[1] "list"
```

"resultat" is a "list" object. It contains two vectors: the key values ($**key**) {1, 2} and the result for each key ($**val**) {4 even numbers, 6 odd numbers}.

We can obtain the keys and the values by using the functions **keys**() and **values**():

```
#keys
print(keys(resultat))

#values: results
print(values(resultat))
```

We have the following output:

```
> #affichage des clés
> print(keys(resultat))
[1] 1 2
>
> #affichage des valeurs calculées
> print(values(resultat))
[1] 4 6
```

3.2 Tracing the execution of MAPREDUCE

In order to track and understand the nature of each step, we add several **print**() procedure in the map and the reduce functions.

```
#map
map_valeurs <- function(., v){
  #print the input vector v
  print("map") ; print(v)
  #key
  cle <- ifelse (v %% 2 == 0, 1, 2)
  #return key and v
  return(keyval(cle,v))
}

#reduce
reduce_valeurs <- function(k, v){
  #print the vector v (subset of the initial vector)
  print("reduce") ; print(k) ; print(v)
  #length of v
  nb <- length(v)
  #return key and result
  return(keyval(k, nb))
}
```

R displays the following output:

```
[1] "map"
 [1]  2  6 67 85  7  9  4 21 78 45
[1] "reduce"
[1] 1
[1]  2  6  4 78
[1] "reduce"
[1] 2
[1] 67 85  7  9 21 45
```

The **map**() function is called only once. The entire vector is processed. The **reduce**() function is called twice : for each key item, we observe the corresponding data subset.

4 Processing a data frame

We want to calculate the sum of the squared residuals (SSR) for a one-way analysis of variance (ANOVA) in this section. The originality here lies in the manipulation and the transmission of a data frame to nodes. The above procedure is altogether transposable to this new configuration. This is the data frame now which will be subdivided into several parts.

Data preparation. We create the dataset as follows:

```
#group membership of the individuals
y <- factor(c(1,1,2,1,2,3,1,2,3,2,1,1,2,3,3))
#values of the response variable
x <- c(0.2,0.65,0.8,0.7,0.85,0.78,1.6,0.7,1.2,1.1,0.4,0.7,0.6,1.7,0.15)
#create a data frame from y and x
don <- data.frame(cbind(y,x))
```

Here is the "don" data table (**data.frame** object):

y	1	1	2	1	2	3	1	2	3	2	1	1	2	3	3
x	0.2	0.65	0.8	0.7	0.85	0.78	1.6	0.7	1.2	1.1	0.4	0.7	0.6	1.7	0.15

Again, we must define the map() and the reduce() procedures.

MAP. Y is a categorical variable, it indicates the group membership. We use it directly for defining the key items.

```
#map
map_ssq <- function(., v){
  #the column y is the key
  cle <- v$y
  #return key and the entire data frame
  return(keyval(cle,v))
}
```

This is the data frame object that the function returns with the key using the **keyval**() function.

REDUCE. The initial data frame is subdivided is subsets defined by Y. We calculate the sum of squares within each group. "v" is a part the "don" data frame here. We have all the variables but only a part of the rows.

```
#reduce - calcul
reduce_ssq <- function(k,v){
  #counting the number of row of the data frame
  n <- nrow(v)
  #calculate the sum of squares using the column 'x' of data.frame
  ssq <- (n-1) * var(v$x)
  #return key and result of calculation
```

```
  return(keyval(k,ssq))
}
```

“v” is a data frame object, we use nrow() and not length() to obtain the number of rows. We use the “$” operator to read the column “x”. The function returns the key and a scalar value.

Calculation. We call the **mapreduce**() procedure of “rmr2”.

```
#rmr2 format
don.dfs <- to.dfs(don)

#mapreduce
calcul <- mapreduce(input=don.dfs,map=map_ssq,reduce=reduce_ssq)

#retrieve the result
resultat <- from.dfs(calcul)
print(resultat)
```

We obtain the sum of squares within each group which are identified by the key item.

```
$key
[1] 1 2 3

$val
[1] 1.152083 0.142000 1.293675
```

We calculate the sum to obtain the residual sum of squares.

```
#SSR
ssr <- sum(resultat$val)
print(ssr)
```

Nous have **SSR = 2.587758**.

We get the same result with the AOV procedure of R.

```
> #contrôle
> print(aov(x ~ y))
Call:
   aov(formula = x ~ y)

Terms:
                       y Residuals
Sum of Squares  0.149015  2.587758
Deg. of Freedom        2        12

Residual standard error: 0.4643776
Estimated effects may be unbalanced
```

Tracing the execution. We add print() command in the reduce() function, we can see the part of the dataset used in each node…

```
#reduce
reduce_ssq <- function(k,v){
  #print the subset of the data frame used
  print("reduce") ; print(v)
  #number of row of the data frame
  n <- nrow(v)
  # calculate the sum of squares
  ssq <- (n-1) * var(v$x)
  #return the key and the result (value)
  return(keyval(k,ssq))
}
```

We note that the function is called three times for the three subsets of the data frame.

```
[1] "reduce"
   y    x
1  1 0.20
2  1 0.65
4  1 0.70
7  1 1.60
11 1 0.40
12 1 0.70
```

reduce(1)

```
[1] "reduce"
   y    x
3  2 0.80
5  2 0.85
8  2 0.70
10 2 1.10
13 2 0.60
```

reduce(2)

```
[1] "reduce"
   y    x
6  3 0.78
9  3 1.20
14 3 1.70
15 3 0.15
```

reduce(3)

5 Linear Least Squares

The linear least squares of Hugh Develin uses only one key value because the dataset is passed to the mapper in chunks of complete rows[9]. The map function is called several times.

In this section, we use the map() function to create subsets of dataset. Thus, the map function is called once, and this is the reduce function which is called several times.

We use K = 2 subsets, but the extension to any number of distinct keys is straightforward. Starting from the program proposed in this section, we must modify only the map() procedure. The reduce() function and the consolidation of the results do not need to be modified.

[9] https://github.com/RevolutionAnalytics/rmr2/blob/master/docs/tutorial.md

We will also use this example to go further in handling the results from the reduce() function. Instead of returning a scalar value in the output of the reduce() function, we will output a more sophisticated structure. We can evaluate the flexibility of the tool in this context.

Dataset. We use the **mtcars** dataset.

```
#data(mtcars)
data(mtcars)
print(mtcars)
```

We want to model the relationship between **mpg** (dependant variable) and the other variables.

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

MAP. The map() function randomly subdivides the data frame into two subsets of approximately equal size (number of rows).

```
#map
map_lm <- function(., D){
  #generate random values
  alea <- runif(nrow(D))
  #generate the key by comparing the random value with 0.5
  #we can easily modify here in order
  #to subdivide the dataset into K subsets (K >= 2)
  cle <- ifelse(alea < 0.5, 1, 2)
  #return key and values (data frame)
  return(keyval(cle,D))
}
```

Comment 1: The randomness of the subdivision is not required in the context of the regression. We can select the first n_1 instances for the first subset, and the remaining for the second subset (n_2). Whatever the partition strategy used, we must obtain the same estimated model coefficients at the end of the calculations.

Comment 2: The generalization into K subsets is easy. Thus, the code for reduce() and the consolidation below work regardless of the number of requested nodes. Only the map() procedure must be modified.

REDUCE. Let us look a little on the ordinary least squares (OLS) estimation before describing the reduce() function. The linear model is written as follows:

$$y = Xa + \varepsilon$$

Y is the target variable. X is the matrix corresponding to the explanatory variables (regressors), a first column of 1 is combined to the matrix to account for the regression constant; a is the vector of parameters that we want to estimate; ε is the error term which captures all the other factors, other than the regressors, which influence the target variable.

The OLS (ordinary least squares) estimate [â] of the vector of parameters is obtained with the following formula:

$$\hat{a} = (X^t X)^{-1} X^t y$$

Where X^t is the transpose of X.

We analyze the entries of the matrices in order to understand the strategy used for the subdivision of the calculations. For (X^tX), at the intersection of X_j and X_m, we have:

$$\sum_{i=1}^{n} x_{ij} \times x_{im}$$

Since the terms are additive, we can split the calculations in 2 parts:

$$\sum_{i=1}^{n_1} x_{ij} \times x_{im} + \sum_{i=n_1+1}^{n} x_{ij} \times x_{im}$$

We have the same phenomenon for (X^ty), at the intersection of X_j and y:

$$\sum_{i=1}^{n} x_{ij} \times y_i = \sum_{i=1}^{n_1} x_{ij} \times y_i + \sum_{i=n_1+1}^{n} x_{ij} \times y_i$$

We can easily subdivide the calculations into K parts in view of these properties. We write the reduce() function as follows:

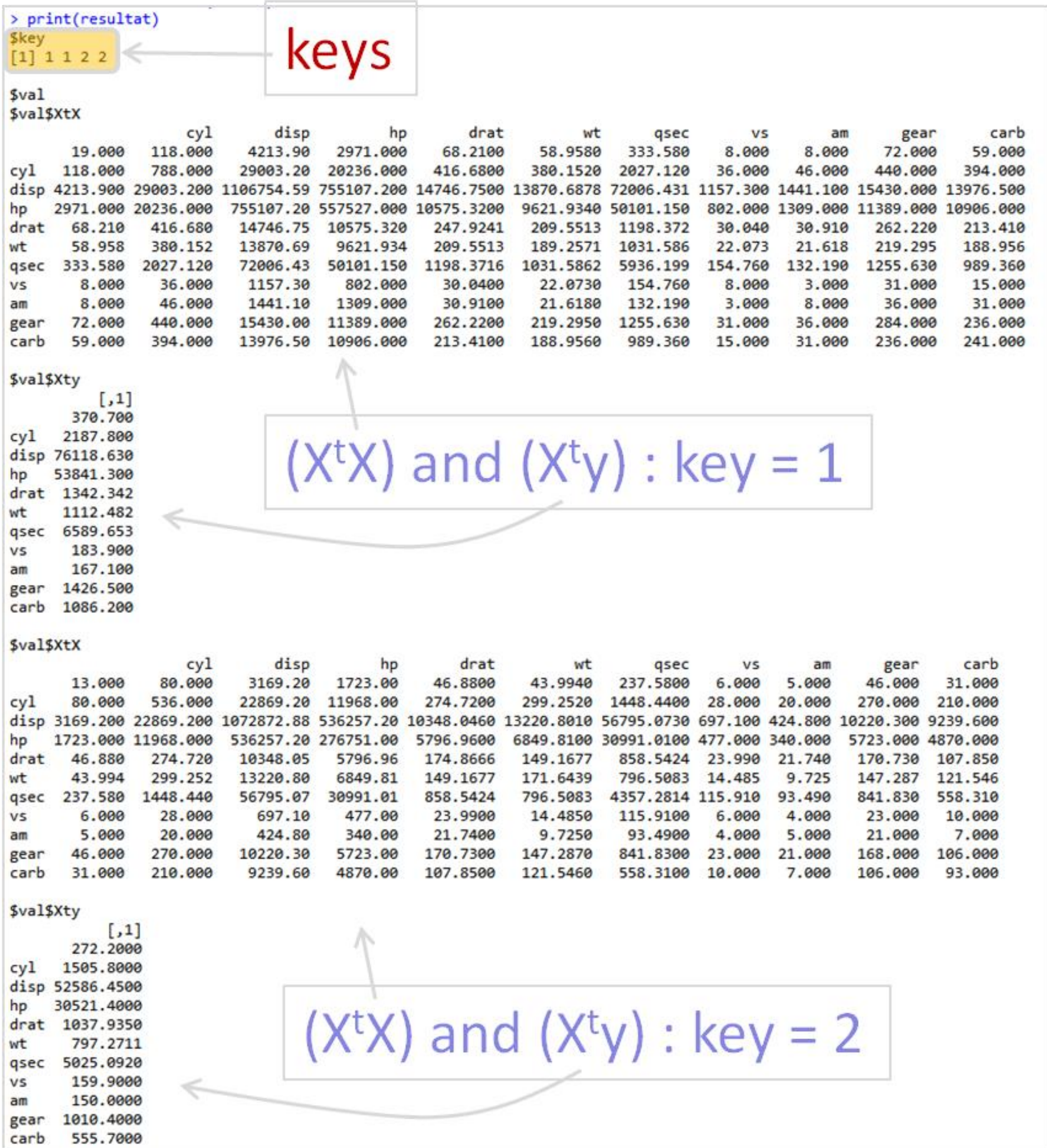
```
#reduce
reduce_lm <- function(k,D){
  #number of rows of the data frame
  n <- nrow(D)
  #target variable
  y <- D$mpg
  #regressors
  X <- as.matrix(D[,-1])
  #add the constant column 1 to the first column
  X <- cbind(rep(1,n),X)
  #calculate XtX
  XtX <- t(X) %*% X
  #calculate Xty
  Xty <- t(X) %*% y
  #set the results into a list
  res <- list(XtX = XtX, Xty = Xty)
  #return key and the list
  return(keyval(k,res))
}
```

The new subtlety is that we use a list to return the two matrices (X^tX) and (X^ty). We must be very attentive when we should consolidate the results to form the corresponding global matrices.

Calculations. We use the mapreduce() function to launch the analysis...

```
#format rmr2
don.dfs <- to.dfs(mtcars)
#mapreduce
calcul <- mapreduce(input=don.dfs,map=map_lm,reduce=reduce_lm)
#récupération
resultat <- from.dfs(calcul)
print(resultat)
```

Let us see the details of the "resultat" object:



```
> print(resultat)
$key
[1] 1 1 2 2

$val
$val$XtX
              cyl      disp        hp       drat        wt      qsec      vs       am      gear      carb
       19.000  118.000   4213.90  2971.000   68.2100   58.9580  333.580   8.000    8.000    72.000    59.000
cyl   118.000  788.000  29003.20 20236.000  416.6800  380.1520 2027.120  36.000   46.000   440.000   394.000
disp 4213.900 29003.200 1106754.59 755107.200 14746.7500 13870.6878 72006.431 1157.300 1441.100 15430.000 13976.500
hp   2971.000 20236.000 755107.20 557527.000 10575.3200 9621.9340 50101.150 802.000 1309.000 11389.000 10906.000
drat   68.210  416.680  14746.75 10575.320  247.9241  209.5513 1198.372  30.040   30.910   262.220   213.410
wt     58.958  380.152  13870.69  9621.934  209.5513  189.2571 1031.586  22.073   21.618   219.295   188.956
qsec  333.580 2027.120  72006.43 50101.150 1198.3716 1031.5862 5936.199 154.760  132.190  1255.630   989.360
vs      8.000   36.000   1157.30   802.000   30.0400   22.0730  154.760   8.000    3.000    31.000    15.000
am      8.000   46.000   1441.10  1309.000   30.9100   21.6180  132.190   3.000    8.000    36.000    31.000
gear   72.000  440.000  15430.00 11389.000  262.2200  219.2950 1255.630  31.000   36.000   284.000   236.000
carb   59.000  394.000  13976.50 10906.000  213.4100  188.9560  989.360  15.000   31.000   236.000   241.000

$val$Xty
          [,1]
       370.700
cyl   2187.800
disp 76118.630
hp   53841.300
drat  1342.342
wt    1112.482
qsec  6589.653
vs     183.900
am     167.100
gear  1426.500
carb  1086.200

$val$XtX
              cyl      disp        hp       drat        wt      qsec      vs      am      gear     carb
       13.000   80.000   3169.20   1723.00   46.8800   43.9940  237.5800   6.000   5.000    46.000   31.000
cyl    80.000  536.000  22869.20  11968.00  274.7200  299.2520 1448.4400  28.000  20.000   270.000  210.000
disp 3169.200 22869.200 1072872.88 536257.20 10348.0460 13220.8010 56795.0730 697.100 424.800 10220.300 9239.600
hp   1723.000 11968.000 536257.20 276751.00 5796.9600 6849.8100 30991.0100 477.000 340.000 5723.000 4870.000
drat   46.880  274.720  10348.05   5796.96  174.8666  149.1677  858.5424  23.990  21.740   170.730  107.850
wt     43.994  299.252  13220.80   6849.81  149.1677  171.6439  796.5083  14.485   9.725   147.287  121.546
qsec  237.580 1448.440  56795.07  30991.01  858.5424  796.5083 4357.2814 115.910  93.490   841.830  558.310
vs      6.000   28.000    697.10    477.00   23.9900   14.4850  115.9100   6.000   4.000    23.000   10.000
am      5.000   20.000    424.80    340.00   21.7400    9.7250   93.4900   4.000   5.000    21.000    7.000
gear   46.000  270.000  10220.30   5723.00  170.7300  147.2870  841.8300  23.000  21.000   168.000  106.000
carb   31.000  210.000   9239.60   4870.00  107.8500  121.5460  558.3100  10.000   7.000   106.000   93.000

$val$Xty
           [,1]
       272.2000
cyl   1505.8000
disp 52586.4500
hp   30521.4000
drat  1037.9350
wt     797.2711
qsec  5025.0920
vs     159.9000
am     150.0000
gear  1010.4000
carb   555.7000
```


Into **$key**, we have a vector with the following values (1, 1, 2, 2). We note that each key item is repeated twice because our reduce() function returns two objects in a list [(X^tX) and (X^ty)].

Into **$val**, we have a list structure where the matrices (X^tX) and (X^ty) are followed one another for each key value.

Thus, to form the global matrix (X^tX) [respectively (X^ty)], we must sum the objects (the matrices) at the position (1, 3) [respectively (2, 4)].

Consolidation of the results. The following consolidation program is operational whatever the number of nodes used (i.e. the number of distinct keys $K \geq 1$).

```
#consolidation

#XtX
MXtX <- matrix(0,nrow=ncol(mtcars),ncol=ncol(mtcars))
for (i in seq(1,length(resultat$val)-1,2)){
  MXtX <- MXtX + resultat$val[[i]]
}
print(MXtX)

#Xty
MXty <- matrix(0,nrow=ncol(mtcars),ncol=1)
for (i in seq(2,length(resultat$val),2)){
  MXty <- MXty + resultat$val[[i]]
}
print(MXty)
```

We obtain the global matrices (X^tX) and (X^ty) :

```
> print(MXtX)
             cyl       disp         hp       drat        wt       qsec       vs       am      gear      carb
      32.000   198.000    7383.10    4694.00   115.0900   102.9520    571.160   14.000   13.000   118.000    90.000
cyl  198.000  1324.000   51872.40   32204.00   691.4000   679.4040   3475.560   64.000   66.000   710.000   604.000
disp 7383.100 51872.400 2179627.47 1291364.40 25094.7960 27091.4888 128801.504 1854.400 1865.900 25650.300 23216.100
hp   4694.000 32204.000 1291364.40  834278.00 16372.2800 16471.7440  81092.160 1279.000 1649.000 17112.000 15776.000
drat  115.090   691.400   25094.80   16372.28   422.7907   358.7190   2056.914   54.030   52.650   432.950   321.260
wt    102.952   679.404   27091.49   16471.74   358.7190   360.9011   1828.095   36.558   31.343   366.582   310.502
qsec  571.160  3475.560  128801.50   81092.16  2056.9140  1828.0946  10293.480  270.670  225.680  2097.460  1547.670
vs     14.000    64.000    1854.40    1279.00    54.0300    36.5580    270.670   14.000    7.000    54.000    25.000
am     13.000    66.000    1865.90    1649.00    52.6500    31.3430    225.680    7.000   13.000    57.000    38.000
gear  118.000   710.000   25650.30   17112.00   432.9500   366.5820   2097.460   54.000   57.000   452.000   342.000
carb   90.000   604.000   23216.10   15776.00   321.2600   310.5020   1547.670   25.000   38.000   342.000   334.000
```

```
> print(MXty)
           [,1]
        642.900
cyl    3693.600
disp 128705.080
hp    84362.700
drat   2380.277
wt     1909.753
qsec  11614.745
vs      343.800
am      317.100
gear   2436.900
carb   1641.900
```

Estimation of the parameters of the model. We get the estimated parameters by using the solve() procedure.

```
#coefficients de la régression
a.chapeau <- solve(MXtX,MXty)
print(a.chapeau)
```

The regression coefficients for the “mtcars” dataset are:

(intercept)

```
> print(a.chapeau)
            [,1]
     12.30337416
cyl  -0.11144048
disp  0.01333524
hp   -0.02148212
drat  0.78711097
wt   -3.71530393
qsec  0.82104075
vs    0.31776281
am    2.52022689
gear  0.65541302
carb -0.19941925
```

Check - The lm() procedure of R. We perform the same regression using the lm() procedure:

```
> #vérification
> print(summary(lm(mpg ~ ., data = mtcars)))

Call:
lm(formula = mpg ~ ., data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max 
-3.4506 -1.6044 -0.1196  1.2193  4.6271 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)  
(Intercept) 12.30337   18.71788   0.657   0.5181  
cyl         -0.11144    1.04502  -0.107   0.9161  
disp         0.01334    0.01786   0.747   0.4635  
hp          -0.02148    0.02177  -0.987   0.3350  
drat         0.78711    1.63537   0.481   0.6353  
wt          -3.71530    1.89441  -1.961   0.0633 .
qsec         0.82104    0.73084   1.123   0.2739  
vs           0.31776    2.10451   0.151   0.8814  
am           2.52023    2.05665   1.225   0.2340  
gear         0.65541    1.49326   0.439   0.6652  
carb        -0.19942    0.82875  -0.241   0.8122  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.65 on 21 degrees of freedom
Multiple R-squared:  0.869, Adjusted R-squared:  0.8066 
F-statistic: 13.93 on 10 and 21 DF,  p-value: 3.793e-07
```

The results are consistent. This is comforting.

6 Conclusion

Simple examples are used in this tutorial to illustrate the MapReduce programming using the package "rmr2" under R. The idea is to subdivide the calculations on a group (cluster) of machines (nodes). Of course, other solutions exist. I had explored the parallelization strategy using other packages[10]. Some of these libraries allow to program an algorithm on a networked computers. We can distribute the calculations on remote machines.

In the treated examples, the map function is used to subdivide the dataset in subsets of data (vector or data frame). The reduce function performs the calculations on these parts of data. We consolidate the results subsequently in order to obtain de global result.

[10] Tanagra tutorial, « Parallel programming in R », october 2013