LINEAR REGRESSION WITH R AND HADOOP

Bogdan OANCEA[*]

Abstract

In this paper we present a way to solve the linear regression model with R and Hadoop using the Rhadoop library. We show how the linear regression model can be solved even for very large models that require special technologies. For storing the data we used Hadoop and for computation we used R. The interface between R and Hadoop is the open source library RHadoop. We present the main features of the Hadoop and R software systems and the way of interconnecting them. We then show how the least squares solution for the linear regression problem could be expressed in terms of map-reduce programming paradigm and how could be implemented using the Rhadoop library.

Keywords: *R, Hadoop, Rhadoop, big data, linear regression.*

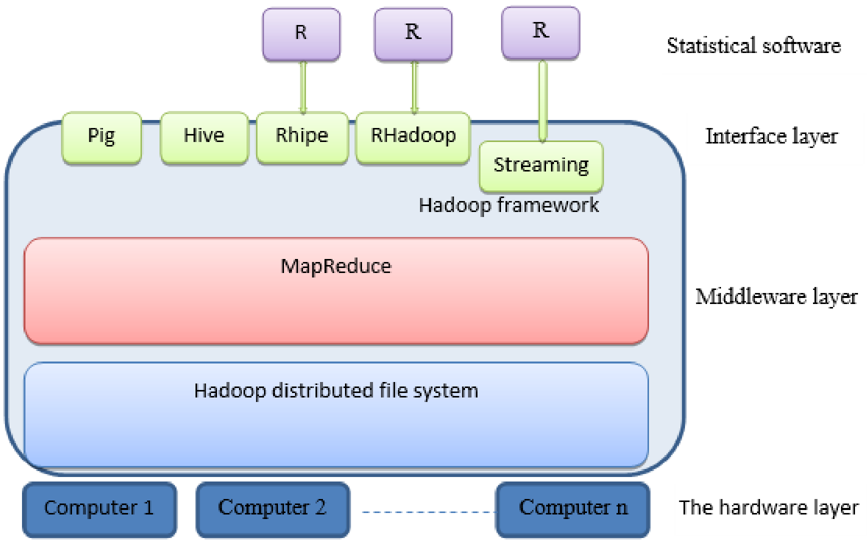
1. Introduction

Hadoop is an open source software whose main purpose is distributed processing of large data sets using computer clusters (White, 2012). Hadoop is developed in Java programming language and is a middleware platform that runs on a cluster of workstations. Applications using Hadoop platform can be developed in Java and in other languages such as R, Ruby or Python. Hadoop can be downloaded from http://hadoop.apache.org. The Hadoop platform users include companies like Yahoo! (Network 2014) or Facebook (Vagata and Wilfong, 2014). Hadoop system consists essentially of:

- Hadoop Distributed File System (HDFS) - a high performance distributed file system;
- Hadoop YARN - a subsystem whose role is scheduling jobs and computer cluster resource management;
- Hadoop Map-Reduce - a system of parallel processing for very large data sets that implements the distributed Map Reduce programming model (Dean, 2004).

Briefly described, Hadoop is a software system that provides its users with a highly reliable distributed file system and a system of analysis and data processing. Hadoop can be installed and run on both clusters with several computers or on clusters with thousands of computers, with a high fault tolerance degree. Currently, Hadoop is a de-facto standard in storing and processing large volumes of data and is used by all major actors in software industry. Hadoop system structure can be observed in Figure 1.

Figure 1. Hadoop and data analysis tools



HDFS file system is based on a client-server architecture. It is a file system with high tolerance to errors and is designed to be run on computers with limited resources. HDFS provides high speed data access, making it ideal for applications that work with large volumes of data, hundreds of GB or TB. HDFS

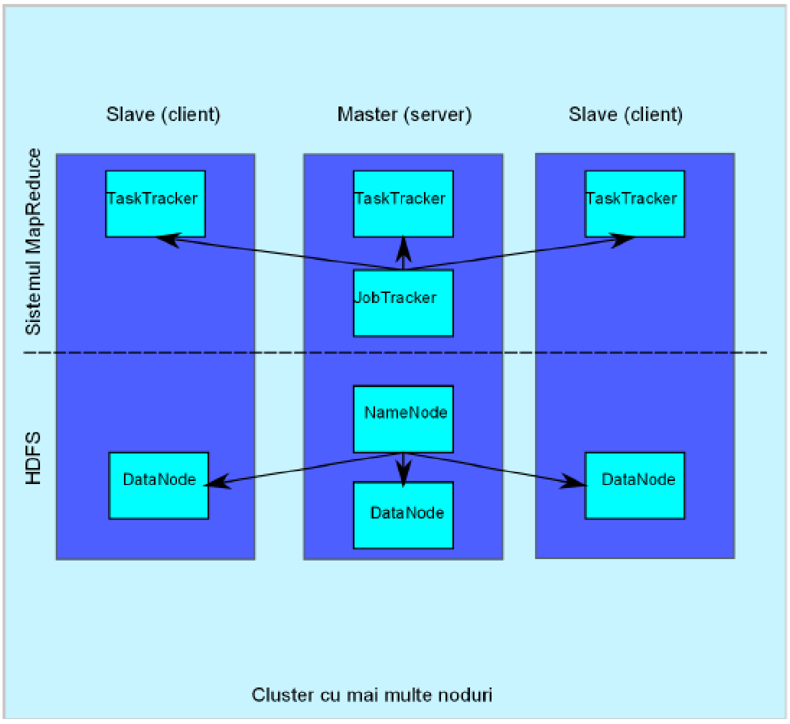
[*] Professor, PhD, Faculty of Economics, "Nicolae Titulescu" University of Bucharest (e-mail: **bogdanoancea@univnt.ro**).

file system is a "append only" system i.e. a file that was created, populated with data and then closed cannot be changed afterwards. This feature simplifies the way to ensure consistency of data files. HDFS provides facilities for bringing processing applications where data is stored as it is more effective to migrate data processing instructions than data. This reduces the data traffic through the computer network interconnection applications thus increasing efficiency

HDFS has a node acting as a server, called NameNode that is run on a master server and one or more DataNode type nodes (acting as clients) managing data storage drives attached to these nodes (computers) from the network (Ryan, 2012). The NameNode aims to manage namespace of the HDFS file system and perform operations of opening, closing or rename files. Files that are saved by HDFS are divided into several blocks of data that are stored by one or more DataNode, responsible for carrying out effective operations of read / write data. Mapping data blocks on nodes is achieved by NameNode. Both NameNode and DataNode application are written in Java and can run on virtually any computer that supports Java.

Hardware fault tolerance is achieved by replicating data blocks on multiple computers. All files are divided into equal-sized data blocks that are then distributed to DataNode. A data block is copied to multiple nodes, so if a node can not operate because of a hardware failure, copies of data are available for other nodes in the network. Over the HDFS distributed file system runs a kernel that that implements the Map Reduce programming model. It consists in a process called JobTracker receiving from clients Map Reduce jobs and schedule them for execution. The JobTracker send processing (jobs) to processes running on the TaskTracker nodes in the computer cluster, trying to keep as close to the data processing that must be processed. If a TaskTracker process does not respond within a certain predetermined time or end with error JobTracker will reschedule the respective processing. Each TaskTracker process starts a Java virtual machine for each job in part to avoid the TaskTracker himself to finish its execution if the job to be executed will lead to termination of the Java virtual machine in case of error. JobTracker and TaskTracker communicate periodically for system status update. Hadoop system structure can be seen in Figure 2.

Figure 2. Software structure of the Hadoop system



R is a free software package for statistics and data visualization (R Core Team, 2013). It is available for several operating systems like UNIX, Windows and MacOS platforms and is the result of the work of many programmers from around the world. R contains facilities for data handling, provides high performance procedures for matrix computations, a large collection of tools for data analysis, graphical functions for data visualization and a straightforward programming language. R comes with about 25 standard packages and many more packages available for download through the CRAN family of Internet sites (http://CRAN.R-project.org). R is used as a computational platform for regular statistics production in many official statistics agencies (Todorov, 2010), (Todorov, 2012). Besides official statistics, it is used in many other sectors like finance, retail, manufacturing, academic research etc., making it a popular tool among statisticians and researchers.

2. Integration between R and Hadoop to process large volumes of data

There is now a large number of R packages or scripts for processing and data analysis. Their use with Hadoop normally would require rewriting them in Java, the natural language for Hadoop, but rewriting activity can lead to many errors. Therefore it is more effective to interface Hadoop ssytem with R so that we they can work with scripts written in R and stored data with Hadoop (Holmes, 2012). Another reason to build an interface between R and Hadoop is that R loads data into memory for processing which can be a serious limitation in terms of the volume of the data.

There are several approaches to integrate R and Hadoop: R and Streaming, Rhipe and RHadoop but in this paper we will present only RHadoop.

RHadoop is an open source project developed by Revolution Analytics (http://www.revolutionanalytics.com/) that provides a client-side integration between R and Hadoop. This allows running Map Reduce jobs within R and consists of a collection of several packages:

- plyrmr - provides plyr like processing functions for structured data type, having capabilities of handling large data sets stored with Hadoop;
- rmr - contains a collection of functions that provide Map Reduce model implementation in R;
- rdfs - is an interface between R and HDFS, providing file management operations in R for data stored in HDFS;
- rhbase - is an interface between R and Hbase, and provides management functions in R for Hbase databases;

RHadoop Installation is very simple, although it depends on other packages. In order to work with R and RHadoop one have to install all depending packages on each DataNode of the Hadoop cluster:

```
> install.packages("RJSONIO")
> install.packages("itertools")
> install.packages("digest")
> install.packages("rJava")
> install.packages("Rcpp")
> install.packages("functional")
> install.packages("reshape2")
> install.packages("plyr")
> install.packages("caTools")
```

rmr package has to be installed from the the archive that contains the source code:

```
>
install.packages("rmr2_3.1.1.tar.gz",repo=NULL,type
="source")
```

The other packages that make up RHadoop rdfs, plyrmr, rhbase, are installed in a similar way.

3. Solving a linear regression using RHadoop

We will illustrate a method of using R with Hadoop to estimate a linear regression model using ordinary least squares method. There are other ways to estimate linear regression models with R and Hadoop, it all depends on the problem to be solved and imagination of the analyst who solve the problem in terms of translating to Map Reduce paradigm which is typical Hadoop (Prajapati, 2013).

A linear regression model takes the following form:

$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathrm{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n,$$

where y_i is the dependent variable and x_i a vector (of dimension p) of regressors that are taken into account (the explanatory variables are independent), and ranges from 1 to n. The n equations can be put in a matrix form:

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_n^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

Or, more simple, like that:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where β vector is the vector of parameters to be estimated.

The ordinary least squares minimizes the sum of squared residues. The calculation formula for estimating vector β is Gujarati (1995) :

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} = \big(\tfrac{1}{n}{\textstyle\sum} \mathbf{x}_i \mathbf{x}_i^{\mathrm{T}}\big)^{-1}\big(\tfrac{1}{n}{\textstyle\sum} \mathbf{x}_i y_i\big).$$

We have to compute a matrix matrix product $X^T X$, than to compute the inverse of the result, $(X^T X)^{-1}$. Next we compute the matrix vector product $X^T y$ and this is multiplied by the intermediate result $(X^T X)^{-1}$. These computations are equivalent with solving a linear system of equations:

$X^T X \beta = X^T y$

where $X^T X$ is the matrix of the linear system, $X^T y$ is the free term and β is the unknown variable. Solving this system is equivalent to the following matrix operation:

$\beta = (X^T X)^{-1} X^T y$

In R there is a predefined function for such problems (solving linear systems):

solve(a, b, ...).

This function takes two parameters: the matrix of the system and free term. In our case we call this function like this:

solve($X^T X$, $X^T y$)

All you have to do is to calculate the transposed matrix X and multiply it by X, then y.

Imagine that we solve the a problem with 20,000 observations (n = 20000) and 15 independent variables

(x_i). In this case the matrix X will have dimensions (20000, 15) and y is the vector of size 20000. Assume that a matrix A (20000, 15) can not be stored in the memory of a single computer and the calculation of transposition and multiplication can't also be performed by a single computer. Instead, the call of :

solve(X^TX, X^Ty)

can be executed easily by one computer. Why? If we try to calculate the dimension of the matrix X^TX we see that we have made a product between two matrices of the following dimensions: (15, 20000) x (20000, 15) = (15, 15). That result is a matrix of size (15,15) and is perfectly feasible that this result will be stored and processed on a single computer. The size of the mltiplication X^Ty is (15, 20000) x (20000, 1) = (15,1). So, and it can be easily stored and processed on a single computer.

What we propose is to use Hadoop to store input data and to perform the two multiplications. The final call solve (X^TX, X^Ty) will be performed classically, on a single computer.

We define the matrix X with random values (following the normal distribution). The number of elements of the matrix is 20000 x 15 = 300,000

```
X <- matrix( rnorm(300000), ncol=15 )
```

We add a new column to the matrix X which will contain the values 1, 2, 3 ... 20000 (number of rows of the matrix X). We will see why we need this new column.

```
X1 <- cbind(1:nrow(X), X)
```

Cbind function performs a column concatenation of the two arguments. The result will be a matrix of size (20000, 16) where the first column contains the values 1, 2, 3, ... nrow (X).

We write the result in the HDFS distributed file system :

```
X1 <- to.dfs(X1)
```

We define now the vector y (the elements of y are all normally distributed random numbers):

```
y <- as.matrix( rnorm( 20000 ))
```

So far we have defined the input data. In a real application they come from files already stored in HDFS.

We define the first map-reduce job in order to calculate X^TX product. First we write first the function map:

```
mapper = function (., Xr) {
Xr = Xr[,-1]
#print(dim(Xr))
keyval(1, list(t(Xr) %*% Xr))
}
```

Here Xr [-1] means all rows and columns of the matrix Xr less column 1. If you remember, column 1 contains the values 1, 2, 3, ... nrow (X).

The map function receives as input data blocks consisting of whole rows of the matrix X. We note such blocks with Xr. If during execution one wants to see the Xr matrix dimmnesions uncomment the line

```
#print (dim (Xr)).
```

On the computer where we tested the script, mapper is called 3 times with the first 7810 rows of the matrix X1, then the next 7811 rows and finally the last 4379 lines.

In this case we are not interested in the key that the mapper receives. The operator %*% achieved multiplication of two matrices. This function will perform smaller submatrix products and will pass the results to the reduce function that will add them up. The function t(X) calculates the transposed of the matrix X. If you analyze the classical algorithm for multiplication of two matrices to calculate X^TX you will find tha onet can form smaller matrix (m rows of the original matrix, with m <n, where n is the total number rows) that you multiply and then assemble the partial results. Therefore, the reducer function will be responsible to collect the partial results issued by the mapper:

```
reducer = function(., Y) {
   keyval(1, list(Reduce('+', Y))
}
```

For details on the function *Reduce* typie help (Reduce) in an R console. This function is similar wtih its counterparts from the LISP language.

So we will have:

```
XtX <- values(
          from.dfs(
mapreduce(
input = X1,
map = mapper,
reduce = reducer,
combine = T
)
)
)[[1]]
```

The *mapreduce* function will write the result as a pair (key, value) in HDFS where it is accessed via from.dfs (...). Function values (...) extracts only the values of pairs (key, value).

We will compute X^Ty in a similar manner:

```
mapper2 = function (., Xr) {
   yr = y[Xr[,1],]
Xr = Xr[,-1]
keyval(1, list(t(Xr) %*% yr))
}
```

yr = y[Xr[,1],] retains from the vector y only the corresponding elements of the lines of matrix Xr. Hadoop will call the mapper with blocks of rows of the original matrix X and we need to select the same rows from y. For this we have introduced new column in X (see instructions X1 <- cbind (1: nrow (X), X)).

```
Xty <- values(
          from.dfs(
mapreduce(
input = X1,
map = mapper2,
reduce = reducer,
combine = T
)
)
)[[1]]
```

We used the parameter combine = T to combine all pairs (key, value) because mapper emits a single key (1).

Finally we call solve:

```
solve(XtX, Xty)
```

The whole R script tha solves the linear regression problem is given in figure 3.

Figure 3. An R script that solves the linear regresion model using Rhadoop

```
#!/usr/bin/Rscript
library(rmr2)
X <- matrix(rnorm(300000), ncol=15)
X1 <- cbind(1:nrow(X), X)
X1 <- to.dfs(X1)
y <- as.matrix(rnorm(20000))

mapper = function (., Xr) {
   Xr <- Xr[,-1]
   #print(dim(Xr))
   keyval(1, list(t(Xr) %*% Xr))
}

reducer = function(., A) {
   keyval(1, list(Reduce('+', A)))
}

mapper2 = function (., Xr) {
   yr <- y[Xr[,1],]
   Xr <- Xr[,-1]
   keyval(1, list(t(Xr) %*% yr))
}

XtX <- values(
   from.dfs(
    mapreduce(
       input = X1,
       map = mapper,
       reduce = reducer,
       combine = T
    )
   )
)[[1]]

Xty <- values(
   from.dfs(
    mapreduce(
       input = X1,
       map = mapper2,
       reduce = reducer,
       combine = T
    )
   )
)[[1]]

beta <- solve(XtX, Xty)
beta
```

4. Conclusions

One of the software tools successfully used for storage and processing of big data sets on clusters of commodity hardware is Hadoop that have become a de-facto standard in big data storage. On the other hand, R is currently used on a large scale for data processing. Interfacing Hadoop and R seems to be the future of big data processing. In this paper we showed how we can solve a linear regression probem using Hadoop and R for very large problems. The algorithm for solving the problem of linear regression has been transformed so that it can use the map-reduce programming model which is psecific to Hadoop.

References

- Dean, J. and S. Ghemawat (2004). Mapreduce: Simplified data processing on large clusters. In OSDI'04, 6th Symposium on Operating Systems Design and Implementation, pp. 137–150. USENIX, in cooperation with ACM SIGOPS.
- Gujarati, D. N. (1995). Basic Econometrics (Third ed.). McGraw Hill.
- Holmes, A. (2012). Hadoop in practice. Manning Publications, New Jersey.
- Network, Y. D. (2014). Hadoop at yahoo! use!R 2007 Conference, Iowa State University. http://developer.yahoo.com/hadoop/.
- Prajapati, V. (2013). Big Data Analytics with R and Hadoop. Packt Publishing.
- White, T. (2012). Hadoop: The Definitive Guide, 3rd Edition. O'Reilly Media.
- R Core Team, (2013), An Introduction to R, available at http://www.r-project.org/, accessed on 25th March 2014.
- Ryan, A., (2012), Under the Hood: Hadoop Distributed Filesystem reliability with Namenode and Avatarnode, available at http://www.facebook.com/notes/facebook-engineering/under-the-hood-hadoop-distributed-filesystem-reliability-with-namenode-and-avata/10150888759153920, last accessed on 25th February, 2015.
- Vagata, P. şi K. Wilfong (2014). Scaling the facebook data warehouse to 300 PB. Available as
- https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb.
- Todorov, V. and M. Templ, (2012), R in the statistical office: Part 2, Development, policy, statistics and research branch working paper 1/2012., United Nations Industrial Development, 2012.
- Todorov, V., (2010), R in the statistical office: The UNIDO experience. Working Paper 03/2010, United Nations Industrial Development. Available at: http://www.unido.org/fileadmin/user_media/Services/Research_and_Statistics/statistics/WP/WP_2010_03.pdf, accessed on 25th February 2015.