

Big data and open-source computation solutions, opportunities and challenges for marketing scientists. Applications to customer base predictive modeling using RFM variables.

Michel CALCIU
Université Lille 1, RIME, France
michel.calciu@iae.univ-lille1.fr

Jean-Louis MOULINS
Université de la Méditerranée, CRETLOG, France
jean-louis.moulins@univ-amu.fr

Francis SALERNO
Université Lille 1, LEM, France
francis.salerno@iae.univ-lille1.fr

Abstract

Marketing researchers and analysts, confined in classical transactional marketing paradigms where customer knowledge was limited to sample based surveys and panels, have somehow been overwhelmed by the avalanche of behavioral data coming from new digital and relationship marketing techniques. This occurred up to a point where a part of what belonged to their core competencies has been overtaken by computer scientists.

The relatively recent open-source solutions that form an ecosystem around the most elegant statistical system, R and the distributed computation system Hadoop (some kind of Linux for computer clusters) democratize BigData calculations and offer excellent opportunities to marketing analysts to operate huge computing factories. An illustration of the implementation of the evoked solutions will be presented, applications to customer data base predictive modeling using RFM variables will be developed and performance gains will be tested.

Key words: Big data, Map Reduce, HPC, predictive modeling, RFM

Introduction

The fascination with the power of computers has often “inflamed” human imagination. In early stages of computers' evolution they were “given” unimaginable powers as in Fredrik Brown's “Answer” a short short story from the sixties (see frame below)

Dwar Ev ceremoniously soldered the final connection ... that would connect, all at once, all of the monster computing machines of all the populated planets in the universe ...
He turned to face the machine. "Is there a God?" The mighty voice answered without hesitation, without the clicking of a single relay. "Yes, now there is a God."
Sudden fear flashed on the face of Dwar Ev. He leaped to grab the switch. A bolt of lightning from the cloudless sky struck him down and fused the switch shut. (Fredric Brown, "Answer")

As computing became popular due to the diffusion of personal computers and office automation software, the « beast » appeared as less fearsome and rather useful in solving problems of “human” scale. The monster woke up again when the need to deal with Big Data came into public attention. “The amount of data in our world has been exploding, and analyzing large data sets -- so-called big data -- will become a key basis of competition, underpinning new waves of productivity growth, innovation, and consumer surplus” Manyika & al. (2011).

Moore's law and the limits of CPU microcosm push to explore CPU macrocosm. After many years of increased computing power through circuit miniaturization, the advances in CPU (Central Processing Unit or microprocessor) *microcosm* at exponential rates, known as More's Law, seem to reach a bottleneck due to physical limits. Also, while processors kept getting faster and faster our datasets got bigger and bigger at an even faster rate. Therefore interest seems to shift progressively towards exploring CPU's *macrocosm* through parallel and distributed computing. It has become usual to have laptop computers with 2 or 4 cores within the same CPU and servers with 8, 32 or more cores are commonplace.

High Performance Computing (HPC) most generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve large problems in science, engineering, or business. It usually deals with supercomputers and/or computer clusters.

Although under continuous development, HPC, dealing with huge amounts of data using supercomputers or computer clusters is regarded by most users, but also by most data scientists and in particular marketing scientists, as something not worth worrying about, or too expensive, or even as a myth or something that appears as unattainable.

Availability of huge computing power or the fascination with a myth becoming reality. Democratized access to immense computing power, often through Internet based cloud-computing¹ generates an new kind of fascination. It is the fascination with the possibility to use such incredible computing power, or more generally speaking, the fascination with myths becoming reality, or with the unfeasible becoming feasible. For academics or scientists, who might get bored, as in doctor Faust's legend, with pushing the limits of knowledge with minuscule steps, getting the opportunity to participate in ground breaking “moments” in science based technologic evolutions is something worth any sacrifice.

Goethe's Faust got fascinated with the possibility to use his knowledge in order to conquer ground from the sea, by building dams, which in those times, the middle-ages, was a wonderful

¹ “Cloud” refers to large Internet services running on tens of thousands of machines (Amazon, Google, Microsoft, etc). “Cloud computing” refers to services by these companies that let external customers rent cycles and storage. Examples are: Amazon EC2: virtual machines at 8.5¢/hour, billed hourly; Amazon S3: storage at 15¢/GB/month; Google AppEngine: free up to a certain quota; Windows Azure: higher-level than EC2, applications use API

achievement for mankind and for the bored scientist, he had become, it was something worth “selling ones sole to the devil”.

Nowadays “bored” scientists might, and probably should, become fascinated with using unleashed computing power to turn Big Data into usable information for the benefits of mankind.

Fascination with power. It should feel good being able to help “shipwrecked” managers who see “Data, data everywhere, and not a byte to use” which is a paraphrase of “Water, water, everywhere, Nor any drop to drink” from Coleridge's *The Rime of the Ancient Mariner*. This feeling or sentiment becomes more powerful when being able, as an individual, to use computer clusters or supercomputers to deal with the data flood, or the data tsunami.

It is this sentiment that was wonderfully exploited in the final scene of the “Doctor Zhivago” movie, a Hollywood masterpiece after Pasternak's famous novel.

Doctor Zhivago's half-brother, who had become a bolshevik and a KGB general, after twenty years of search, finds his half brother's and Lara's daughter and tells her, her parents' love story that had crossed the First World War, the bolshevik revolution and the civil war. The movie renders this epic saga and at the end the girl presents her fiancé and tells he is an operator. When the general asks what he operates, the girl shows proudly through the window a huge dam of an hydro-electric central.

Any computer-literate individual and of course any data scientist can operate, due to technologic advances we are presenting here, huge computer clusters, which is similar in impetuosity with operating huge automated factories, refineries or energetic facilities.

Marketing researchers and analysts, confined in classical transactional marketing paradigms where customer knowledge was limited to sample based surveys and panels, have somehow been overwhelmed by the avalanche of behavioral data coming from new digital and relationship marketing techniques. This occurred up to a point where a part of what belonged to their core competencies has been overtaken by computer scientists.

The relatively recent open-source solutions that form an ecosystem around the most elegant statistical system, R and the distributed computation system Hadoop (some kind of Linux for computer clusters), but also cloud-computing and other technologic advances democratize BigData calculations and offer excellent opportunities to marketing analysts to operate huge computing factories.

How does this all work, how can marketing scientists make use of it, why should they be part of the game, how problems need to be adapted in order to be solved with such enormous capacity and power is dealt with in this paper.

An illustration of the implementation of the evoked solutions will be presented, applications to customer data base predictive modeling using RFM variables will be developed and performance gains will be tested.

What data are Big and Selected open-source solutions

What data are Big, is a rather relative matter. Some authors pretend that a file with more than one million records can be considered as big data. Others indicate sizes of several Tera or even Peta bytes. A less orthodox definition comes from Hadley Wickham who puts that, in traditional analysis, cognition time is longer than computing time while for BigData it is the other way around, the CPU time for computing takes longer than the cognitive process of designing a model.

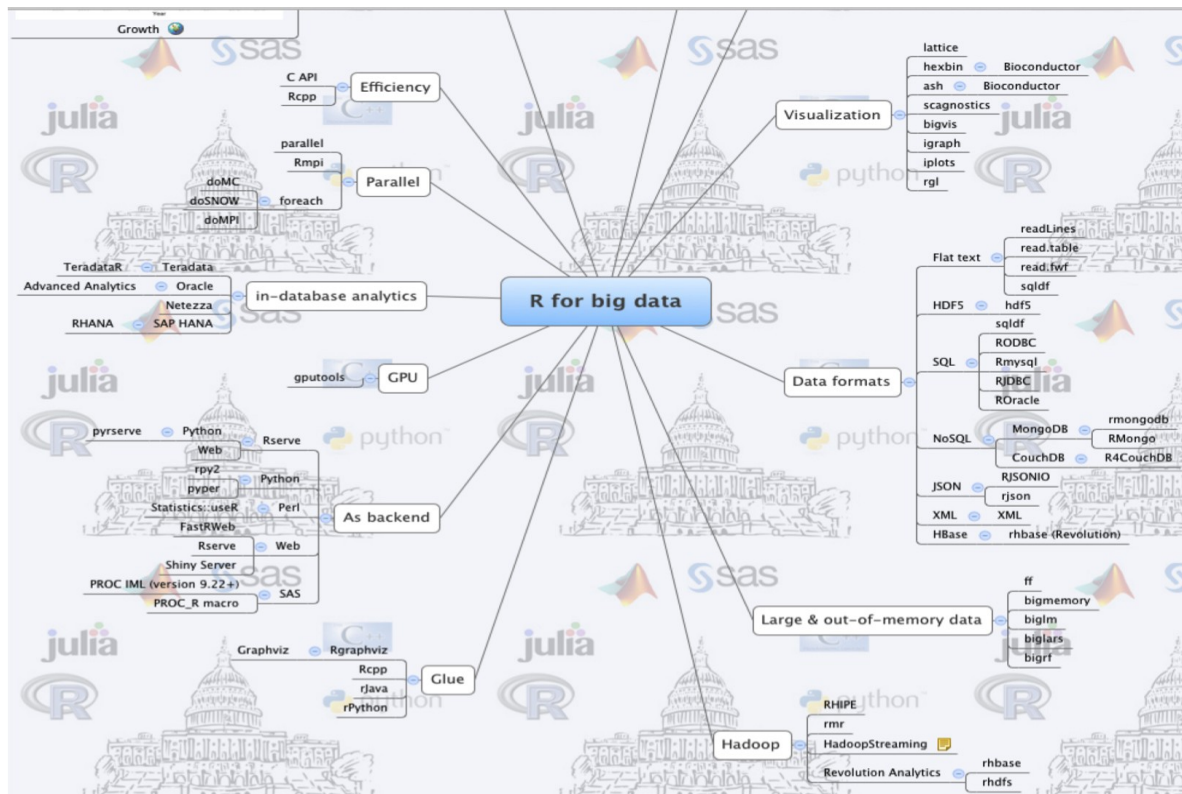
In choosing the tools and solutions to be used by marketing scientists for BigData calculations we insist on actively developed open-source solutions both for the statistical and parallel computing

tasks and concentrate our illustrations and discussions around the statistical system “R” and the distributed computation system Hadoop.

The statistical system R and its BigData Ecosystem

The R statistical system, that progressively becomes the preferred tool for a majority of data analysts and for an increasing number of marketing scientists, had some years ago the reputation of not being able to deal with BigData. This remains true for many other statistical software solutions while it has changed a lot for R. Nowadays R has a series of specialized packages that form an ecosystem for dealing with Big Data (see figure 1)

Figure 1 – BigData Ecosystem in R



Source: <http://www.xmind.net/m/LKF2/> 07/05/2013 last visited 14/09/2015

R ranks in 6th place in the IEEE Spectrum's 2015 list of Top Programming Languages (Cass, 2015) just behind the main general purpose languages like Java, C and Python and jumps 3 places from its 2014 ranking before three other highly respected languages: PHP, Javascript and Ruby. This ranking is impressive for a domain-specific language and demonstrates the importance of big data and advanced data analysis in today's world.

What data are Big for R and how can they be dealt with? Wijfels (2013) suggests three levels as to the size of data. The first or lowest is when data contain less than one million records and can be dealt with using standard R. The second is when data contain between one million and one billion records. They can also be processed in R, but need some additional effort by adopting one or more of the five strategies listed below. The third and highest level is when data contain over one billion records. In this case algorithms can be designed in R and processed with connectors to Hadoop or alternative HPC solutions. The five strategies that can be used to tackle big data with R, as suggested by Bracht (2013) are: 1) Sampling, 2) Bigger Hardware, 3) Storing objects on hard disc, 4) Integration of higher performing programming languages and 5) Alternative interpreters.

Sampling allows to reduce the size of data. While much data is always better than little data, sampling is acceptable, at least if the size of data crosses the one billion record threshold.

Bigger hardware can be a solution when data gets large, as R keeps all objects in memory. One solution is to increase the machine's memory. On 64-bit machines R can address 8 TB of RAM a huge improvement compared to the about 2 GB addressable RAM on 32-bit machines.

Store objects on hard disc and analyze them chunkwise is an alternative that avoids storing data in memory. Chunking leads naturally to parallelization and algorithms need to be explicitly designed to deal with hard disc specific data types. “ff” and “ffbase” are the best known CRAN (open-source) packages following this principle. The “scaleR” package from Revolution R Enterprise is also a popular solution.

Integration of higher performing programming languages like C++ or Java. Parts of the program are moved from R to another language to avoid bottlenecks and performance expensive procedures. The aim is to combine, when dealing with data, the elegance of R with the performance of other languages. It is relatively easy to outsource code from R to these languages using the Rcpp and Rjava packages.

Alternative interpreters might be faster or better suited in some cases than standard R. These are pqR (pretty quick R), Renjin that reimplements the R interpreter in Java, so it can run on the Java Virtual Machine (JVM), TERR a C++ based interpreter created by Tibco. Oracle R uses Intel's mathematic library to achieve higher performance without changing R's core.

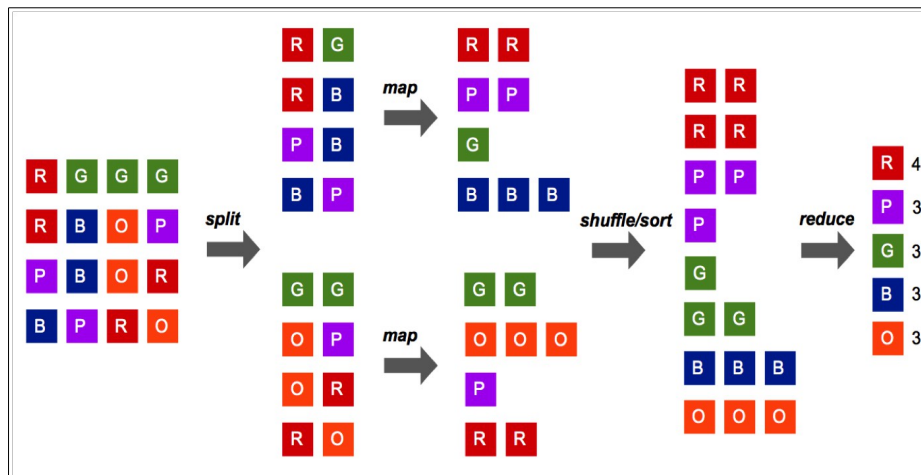
Hiding complexity of Distributed Parallel Computing with MapReduce and Hadoop

While the previously evoked strategies are palliatives when data are not yet too big for R. When data are too big for R, connectors to Hadoop or alternative HPC solutions need to be used in conjunction with R.

MapReduce is a high level programming model and an associated implementation for large scale parallel data processing. It has the merit to hide all complexities of parallel computing on distributed servers from users and to have contributed massively to democratize BigData processing. The name MapReduce originally referred to the proprietary Google technology (Dean and Ghemawat, 2004), but has since become a generic trademark. It is integrated in Apache's Hadoop , an open-source software framework, written in Java, for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware.

It is useful for marketing scientists to have a grasp of MapReduce in order to be able to adapt their models and algorithms to distributed parallel processing of massive datasets. MapReduce is based on the observation that most computations can be expressed in terms of a Map() procedure that enables filtering and sorting and of a Reduce() procedure that performs an aggregating operation like counting, summing etc. Map and Reduce are common Higher-Order Functions in Functional Programming to which the statistical system R belongs. A very simple example is given by White (2012) and illustrated in the figure bellow.

Figure 2 - simple MapReduce example: Colored Square Counter



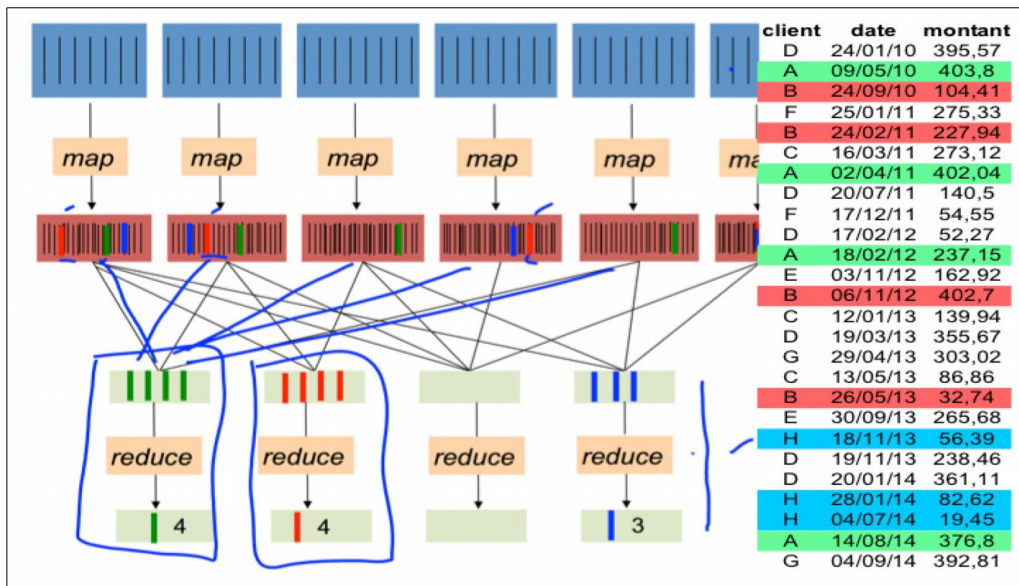
(Source: White , 2012, p. 4)

The input to the MapReduce process in figure 2 is a set of colored squares. The objective is to count the number of squares of each color. The user specifies a *map* function that takes as arguments a key/value pair and returns a set of intermediate key/value pairs. Here the key is the letter indicating the color and the value is one. The *reduce* function merges all intermediate values associated with the same intermediate key. Here it would be either the sum or the length of the vector containing those intermediate values. The remainder processing is handled by the software system implementing the MapReduce programming model as shown in figure 2. “The MapReduce system first reads the input file and splits it into multiple pieces, here two. These splits are then processed by multiple map programs running in parallel on the nodes of the cluster and group the data in a split by color. The system then takes the output from each map program and merges (shuffle/sort) the results for input to the reduce program, which calculates the sum of the values of the squares for each color. In this example, only one copy of the reduce program is used, but there may be more in practice. To optimize performance, programmers can provide their own shuffle/sort program and can also deploy a combiner that combines local map output files to reduce the number of output files that have to be remotely accessed across the cluster by the shuffle/sort step.” (White, 2012).

Using MapReduce in simple BigData customer management calculations (RFM)

The MapReduce approach to counting squares by color is very similar to counting transactions by customer or customer transactions Frequency. Frequency (F) together with Recency (R) and Monetary (M) are fundamental variables for behavioral customer segmentation and targeting.

Figure 3 - MapReduce example: RFM calculation on a customer base



Source: adapted from Howe B. eScience Institute, U. Washington, 2013

In the example above we have customer transactions instead of color squares. Each transaction, as can be seen on the right hand side of Figure 3, is a record that contains the customer's identifier, the transaction date and the amount spent on that occasion. The file we are using can be considered as BigData as it contains 343766402 transactions (records) recorded during 78 weeks from 6326658 customers. For confidentiality reasons the source of the data cannot be disclosed.

So for the *Frequency*, the *map* function returns as intermediate key/value pair the identifier of the customer and the value one. The reduce function will be, as in the previous example, either the sum or the length of the vector containing those intermediate values by key.

For the *Recency* which, in this case, is the date of the last transaction by customer, the map function should return as a value in the key/value pair the date of transaction, while the reduce function will merge the maximum date by customer.

For the *Monetary* which could be the total or average amount spent by each customer, the map function should return as a value in the key/value pair the amount spent by transaction, while the reduce function would merge by customer either the sum or the average of the amount spent.

As previously, if the data were too big and needed to be processed on a computer cluster, the MapReduce process (see Figure 3) would read the input file and split it into multiple chunks. These chunks or splits are then processed by multiple map programs running in parallel on the nodes of the cluster and group the data in a chunk by customer. The system then takes the output from each map program and merges (shuffle/sort) the results for input to the reduce program, which calculates per customer, the length (or sum of values) of the vector for Frequency, the maximum value for Recency and the sum or the average value for Monetary.

As the data we use contain about 344 million records they can be considered “middle sized” for R, as to Wiffel's (2013) categorization that has been mentioned above. This means that they can be dealt with using one of the five strategies mentioned above, in our case the strategy applied is using bigger hardware, that is a MacBookPro computer with a rather powerful double core Intel I7 CPU and 16 GB (Giga Bytes) of memory. Bigger memory was necessary as the object size of the data file after being read into the computer's memory was 5.5 GB.

As R is also a Functional Programming Language, Map and Reduce functions and/or equivalent higher-order functions can be used also on only one machine. In this case a function that applies the

mapreduce approach that we used is `tapply()`. Its syntax is `tapply(value, key, FUN)` where `value` and `key` can be the outputs of a map function while `FUN` is a merging reduce function like `sum`, `mean`, `length` etc.

There from computing the customer frequency, `F`, using a MapReduce approach (but not the MapReduce process) on a single machine consists in using `tapply(rep(1,nrow(X)), client, sum)` or `tapply(rep(1,nrow(X)), client, length)`. In this case each record (transaction) is “mapped” being given the *value* 1 (one) and the customer's identifier as a key. The reduce function can be the `sum` or the `length` as the value vector contains only ones.

For the amount spent by customer or the monetary (`M`) we use `tapply(montant, client, sum)` or `tapply(montant, client, mean)`. Each record is mapped being given the amount spent per transaction as value and the customer's identifier as a key. The reduce function is be the `sum` or the `mean` amount spent.

For the date of the last transaction or recency (`R`) we use `tapply(date, client, max)`. Each record is mapped being given the date of each transaction as value and the customer's identifier as a key. The reduce function is the maximum date.

The system duration of each of these map and reduce calculations, that is the time it took to obtain each RFM value per customer, was less than 7 minutes. This rather good performance is due to the speed of the CPU and to the fact that the whole data object could be loaded into memory.

Implementing a MapReduce process with R and Hadoop. When data are too big to fit in one computer's memory a MapReduce process on a cluster of computers needs to be put in place. MapReduce compared to standard HPC approaches is a “share nothing” process.

The MapReduce process can be implemented after installing the open-source HADOOP system on each computer of the cluster. There is a series of R packages that implement the so called RHadoop solution that are freely available from Revolution Analytics. They keep all parallel computing complexities transparent to users and are therefore a highly recommendable to marketing scientists.

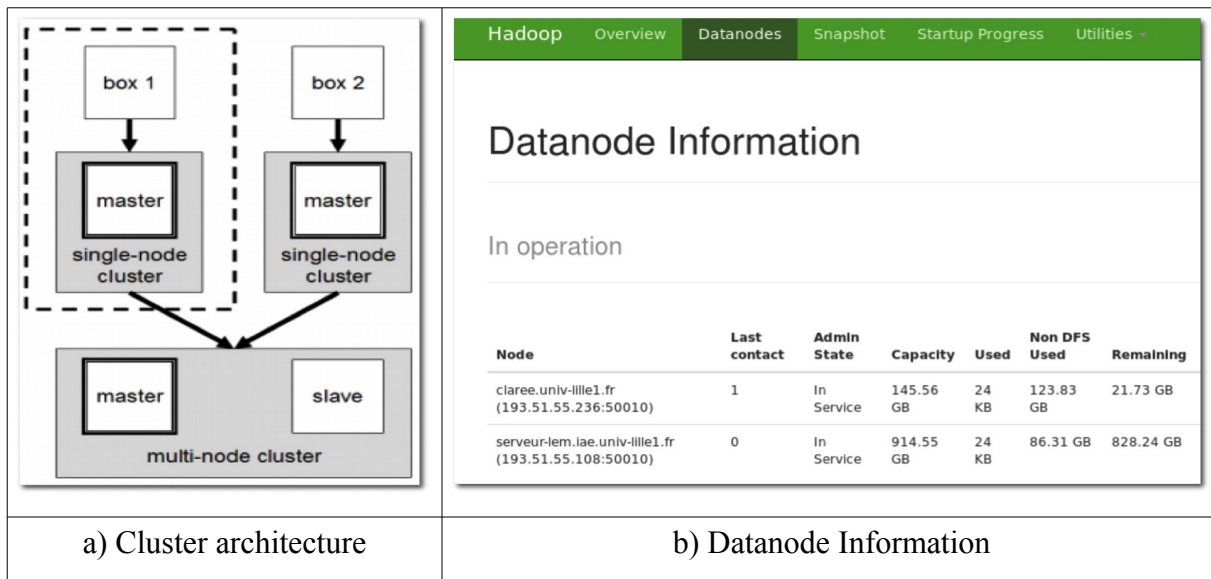
The listing below shows how “simple” it is to launch a MapReduce process to compute customer Frequency over a computer cluster using two Rhadoop packages over a BigData file containing customer transactions.

```
library(hdfs)
library(rmr2)
hdfs.data="transactions.csv"
map <- function(.,v) { keyval(as.numeric(v[[1]][-1]), 1)}
reduce <- function(k, v) { keyval(k, length(v)) }
mapreduce(input=hdfs.data, input.format=make.input.format("csv", sep=";"), map=map, reduce=reduce)
```

The listing loads two Rhadoop packages, defines the map and reduce functions as before using key-value pairs and launches the MapReduce process over a cluster of computers.

The duration of the calculations depends on several aspects: the number of nodes (computers) in the cluster, their CPU speed and Memory. As the calculations take place on a computer cluster over one network there are practically no limits for the size of the data. We have installed Hadoop for demonstration and testing purposes on a small cluster of two Linux computers as can be seen from figure and have run the listing above.

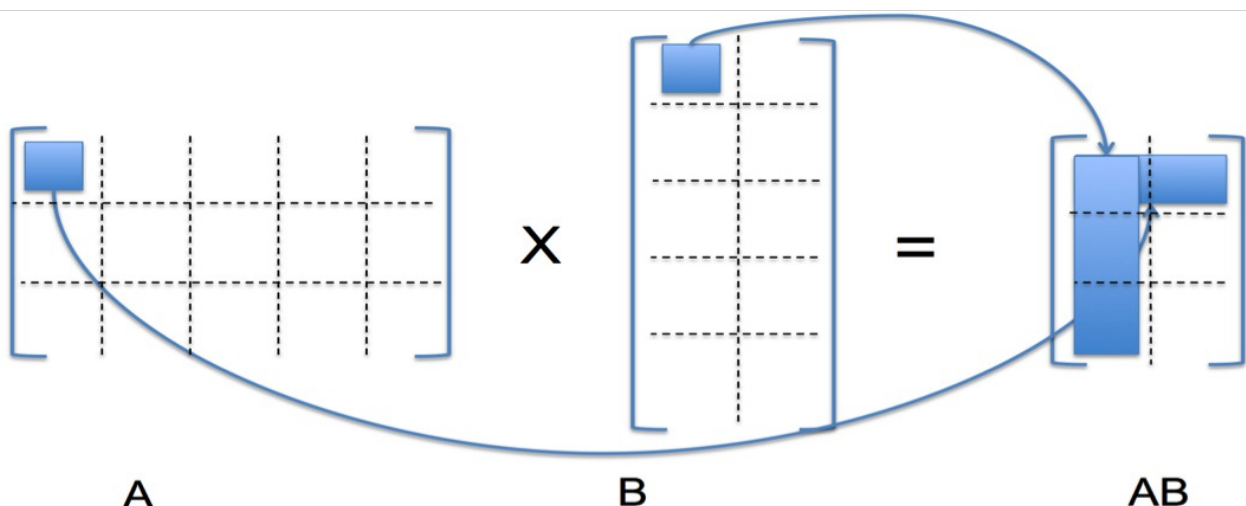
Figure 4 - A test Hadoop computer cluster and datanode information



Using MapReduce in more advanced marketing science problems

While computing customer RFM variables in marketing are quite straight forward applications of the MapReduce approach, more sophisticated marketing science models need to and many can be adapted for MapReduce. Many models and calculations used in marketing science and data analysis use linear algebra calculations. One very important calculation that needs to be adapted to the MapReduce approach is matrix multiplication. Multivariate models like Linear Regression, Factor Analysis or Discriminant Analysis use computationally more sophisticated algorithms over a summary often symmetric matrix of rather small dimensions given by the number of variables. This small matrix is obtained by applying matrix multiplication to one or two BigData data matrixes resulting from customer recordings, observations or declarations. In those cases MapReduce needs only be applied to the multiplication of those BigData matrixes.

Figure 5 - Organizing matrix multiplication for MapReduce



As can be seen from figure 6 in the resulting matrix AB each cell is a sum of products of all cells of the corresponding line in the first matrix (A) with all cells of the corresponding column in the second matrix (B). In terms of MapReduce all these cells have the same final reduce key, that corresponds to the position (line and column index) of the given cell in the resulting matrix AB. The

resulting matrix AB has dimension $I \times K$ where I is the number of rows of matrix A and K is the number of columns of matrix B. Also each cell (i,j) in matrix A is used to compute all K cells that correspond to its row (i) in the resulting matrix, that is as many cells as there are columns in matrix B. Each cell in matrix B is used to compute all I cells that correspond to its column (k) in the resulting matrix, that is as many cells as there are rows in matrix A.

Therefore in the map phase each (i,j) element (or cell) in A needs to be duplicated K times and each duplicate value will have an associated key (i,k) where $k=1..K$ or in a more formalized way:

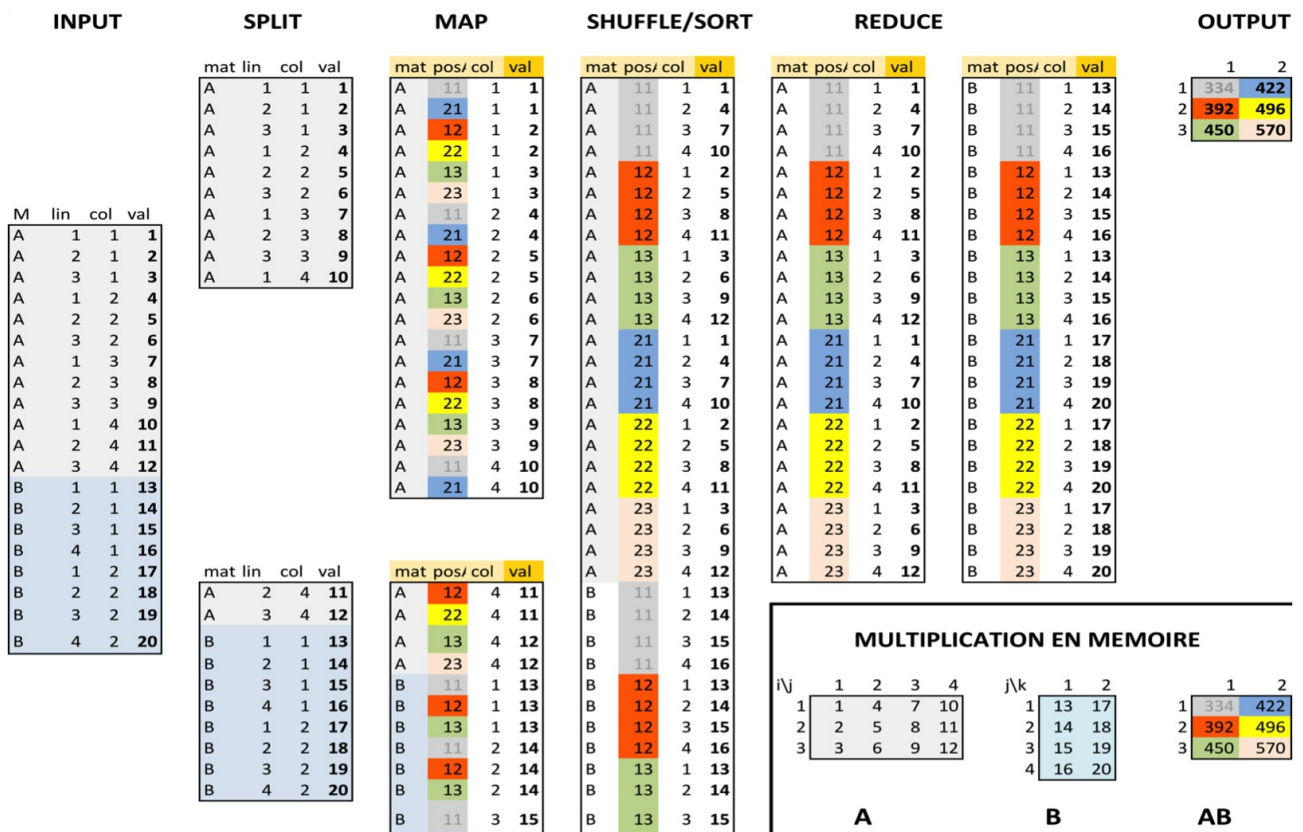
for each (i,j) cell of A $\rightarrow ((i,k), A[i,j]), k=1..K$

Also each (j,k) element (or cell) in B needs to be duplicated I times and each duplicate value will have an associated key (i,k) where $i=1..I$ or in a more formalized way:

for each (j,k) cell of B $\rightarrow ((i,k), B[j,k]), i=1..I$

A reducer for each output cell (i,k) will compute the sum of all j products $(A[i,j] * B[j,k])$.

Figure 6 - MapReduce process example for matrix multiplication



As can be seen in figure 6 the MapReduce process that would correspond to the multiplication in memory of the two small matrixes A and B receives as input file the two matrixes written one cell per record. Each record contains the cell value, position (line and column) and matrix it belongs to. In the split phase the input file is split into two chunks each affected to a different node (computer). The map functions operate on each chunk. They duplicate each cell of A four times and each cell of B three times and associates to them the corresponding (i,k) key and the duplicate number j .

In the shuffle/sort phase all cells are regrouped by the (i,k) key and the reduce phase computes the sum of the product of all duplicate cells having the same (i,k) key ordered by j .

Although this process seems complicated, the task is automated and it has the advantage that there are no limits to the size of the matrixes to be multiplied.

A special case of matrix multiplication that is used in Linear Regression, Factor Analysis or Discriminant Analysis is the multiplication of the transposed matrix with itself. This matrix multiplication has a rather straightforward solution with MapReduce that exploits the fact the while multiplication of matrixes is not commutative the sum of them is. In this latter case the multiplication can be done chunkwise in memory in the map phase and the reduce phase will simply sum up the output of all those chunkwise multiplications. Figure 7 illustrates the MapReduce approach and process in this special case. For each phase both R commands and results are displayed.

Figure 7 - MapReduce approach to $X'X$ matrix multiplication

Input	Split	Map	Reduce
<pre>> X=1:8 > dim(X)=c(4,2) > X # big matrix</pre> <div> <div>[,1]</div> <div>[,2]</div> <div>[1,]</div> <div>[2,]</div> <div>[3,]</div> <div>[4,]</div> <div>1</div> <div>2</div> <div>3</div> <div>4</div> <div>5</div> <div>6</div> <div>7</div> <div>8</div> </div>	<pre>> (lXi = list(X[1:2,], X[3:4,]))</pre> <div> <div>[,1]</div> <div>[,2]</div> <div>[1,]</div> <div>[2,]</div> <div>1</div> <div>2</div> <div>5</div> <div>6</div> </div> <div> <div>[,1]</div> <div>[,2]</div> <div>[1,]</div> <div>[2,]</div> <div>3</div> <div>4</div> <div>7</div> <div>8</div> </div>	<pre>> XtX = function(X)t(X)%*%X > (lXtX=Map(XtX, lXi))</pre> <div> <div>[,1]</div> <div>[,2]</div> <div>[1,]</div> <div>[2,]</div> <div>5</div> <div>17</div> <div>17</div> <div>61</div> </div> <div> <div>[,1]</div> <div>[,2]</div> <div>[1,]</div> <div>[2,]</div> <div>25</div> <div>53</div> <div>53</div> <div>113</div> </div>	<pre>> Reduce("+", lXtX)</pre> <div> <div>[,1]</div> <div>[,2]</div> <div>[1,]</div> <div>[2,]</div> <div>30</div> <div>70</div> <div>70</div> <div>174</div> </div> <pre>> t(X) %*% X</pre> <div> <div>[,1]</div> <div>[,2]</div> <div>[1,]</div> <div>[2,]</div> <div>30</div> <div>70</div> <div>70</div> <div>174</div> </div>

The *input* phase Figure 7 produces a matrix with four rows and two columns to play the role of what can be called the big matrix. In the *split* phase the big matrix is transformed in a list with two chunks the first containing rows one and two and the second rows three and four. In the *map* phase a multiplication function of the transposed matrix with itself is defined and mapped on the chunk list producing a list of chunkwise $X'X$ multiplication results. In the *reduce* phase, as the sum of matrixes is commutative, the reduce function sums all chunkwise multiplication result matrixes and obtains the final result which is strictly identical to the one obtained when the transposed of the big matrix would have been multiplied with itself without being split.

MapReduce's data-parallel programming model hides complexity of distribution and fault tolerance. Its main philosophy is to scale computing power by adding hardware to tackle the size of problems. But also to save hardware, programmer and administration costs. MapReduce is not suitable for all problems, new programming models and frameworks are still being created that build upon these ideas. As we could see computing solutions need to be adapted to MapReduce because conventional programs would not work as the data is split across nodes. A promising solution is DAG (Directed Acyclic Graph) a programming style for distributed systems. It can be seen as an alternative to Map Reduce. While MapReduce has just two steps (map and reduce), DAG can have multiple levels that can form a tree structure and is therefore more flexible with more functions like map, filter, union etc. Also DAG execution is faster due to intermediate results not being written to disk. One of the most plebiscited implementation of DAG is Apache Spark. Its main concept "RDD" Resilient Distributed Dataset is best explained by Zaharia & al. (2012)² from Berkley University, who initiated this approach. Spark takes MapReduce to the next level with less

2 In brief, RDDs are distributed data sets that can stay in memory and fallback to disk gracefully. RDDs if lost can be easily rebuilt using a graph that says how to reconstruct. RDDs are great if you want to keep holding a data set in memory and fire a series of queries - this works better than fetching data from disk every time. Another important RDD concept is that there are two types of things that can be done on an RDD - 1) Transformations like, map, filter than results in another RDD. 2) Actions like count that result in an output. A spark job comprises of a DAG of tasks executing transformations and actions on RDDs.

expensive shuffles in the data processing. With capabilities like in-memory data storage and near real-time processing, the performance can be several times faster than other big data technologies.

Other HPC solutions for marketing scientists

While MapReduce and later approaches completely hide complexities of distributed and parallel computing and therefore are easily implementable as services and accessible to anyone through cloud-computing, there is a whole area of HPC methods and facilities that are also available to marketing scientists as members of academia. Many universities have or participate in computer cluster or grid projects. The authors as members of a university, that ranks fifth in France as computing capacity of its computer cluster³, could use these facilities and a series of R packages in order to test performance gains when doing predictive modeling using RFM variables by varying the number of cores and computers in the cluster. For a detailed list of available R packages one could read the official web page of the CRAN Task View concerning High-Performance and Parallel Computing with R⁴. We introduce a Gross Recording Approach for Parallel Performance Analysis (GRAPPA) and apply it to calculations that estimate several predictive models based upon RFM (Recency, Frequency, Monetary) buying behavior variables. The two data sets being used come from a Fast Moving Consumer Goods retail chain and from a mail-order catalogue. They reflect the repeat buying behavior during several seasons for cohorts of customers. The first dataset has already been used in this paper to illustrate the MapReduce approach to compute RFM variables and shrink the huge transactions table of 344 million records to a smaller but still big customer table with 6.3 million records. We use this table to estimate several predictive models like Linear, Logit, Probit, neural networks or CART classification trees, in order to measure the impact of the RFM variables on purchase incidence. As the file is rather big, the main purpose here is to measure the time it takes to calibrate a model using one core of the CPU, here the MacBookPro Intel i7 microprocessor. The shortest time, 4-5 seconds, has been attained with linear regression and the longest with neural networks took more than 820 seconds or about 14 minutes and 320 iterations to converge. A more detailed discussion on these models and their estimation with RFM variables can be found in Calciu & Salerno (2005).

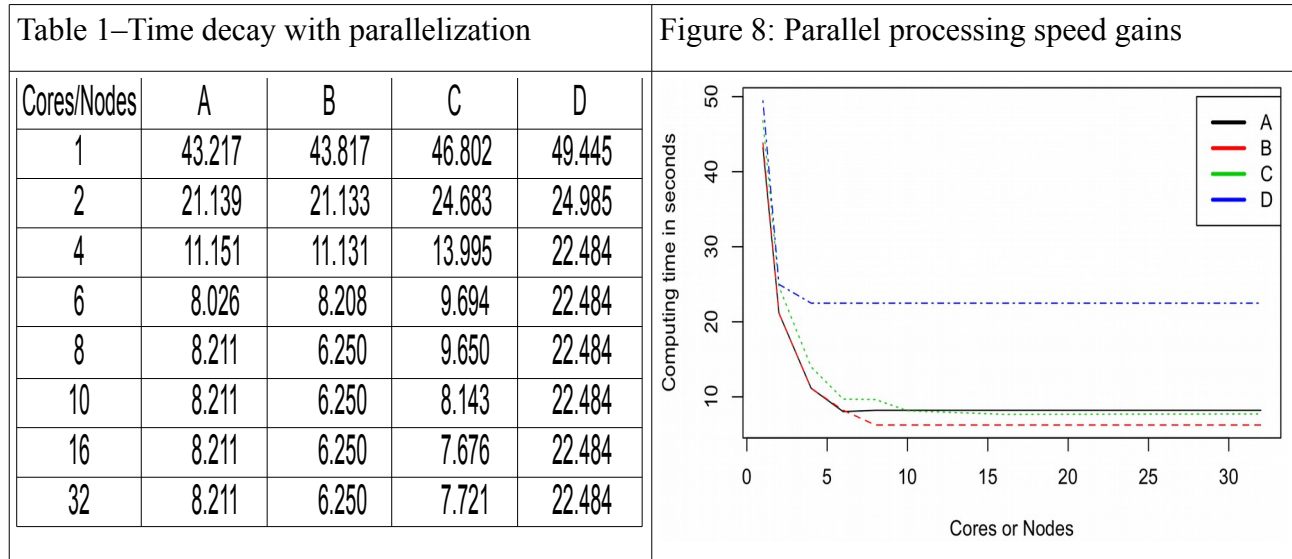
As multiple linear regression could easily be solved by mimicking a MapReduce approach on the same MacBookPro laptop computer, we applied it to compare serial to implicit parallel calculations by varying the number of CPU cores from 2 to 4. Linear regression can be solved with simple multiplications of the matrix, containing the values of independent variables and a first column of ones, called \mathbf{X} and of a vector, containing the values of the dependent variable, called \mathbf{y} . Supposing limited memory, these big tables of 6.3 million records have been split according to the number of cores into 2 to 4 chunks. *Map functions* have been defined to compute $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}'\mathbf{y}$ multiplications chunkwise and applied both serially and simultaneously using implicit parallelism. For the later the *mcapply* function from the R *parallel* package has been used. The *Reduce function* simply exploits the fact that the matrix sum is associative and commutative as we showed in Figure 7 (for details see Appendix). We repeated the serial and parallel calculation on 2, 3 and 4 chunks one hundred times. The paired t-test showed a significant positive difference in computing time between serial and parallel calculations ($t = 40.2006$, $df = 299$, $p\text{-value} < 2.2e-16$). While serial calculations had an average duration of 1.99 seconds, parallel calculations had only 1.44 and the mean of the differences was .51 seconds. The same significant positive difference is observed by analyzing calculations using 2 chunks (0.60855s), 3 chunks (0.21772s) and 4 chunks (0.68903) separately.

The second file with its six thousand records representing the purchase incidence of mail-order customers is much smaller and has been used to perform model leave-one-out cross-validation in order

3 The computer cluster used consists of heterogeneous machines that include 2110 CPU cores and 7168 GPU cores. The cluster has disk capacity of about 156 Tb and a theoretical speed of about 50 Tflops. All machines are connected through a Infiniband 40 Gbps network.

4 <https://cran.r-project.org/web/views/HighPerformanceComputing.html>

to limit overfitting and improve predictive validity. Cross-validation, like bootstrapping is an “embarrassingly parallel” problem and therefore easy to parallelize. Leave-one-out cross-validation in our dataset of 6000 observations involved repeating model fitting 6000 times over training sets obtained by eliminating each time one observation. This repetitive calculation could easily be parallelized both for *implicit* or *multicore parallelism* meaning several cores sharing the same memory and for *cluster parallelism* which combines several computers in a cluster.



A: linear, nodes1/cores >0; B: logistic, nodes1/cores >0; C: logistic, nodes > 0/cores 1; logistic, interactive, node 1/cores >0

The first three columns in table 1 record parallelization performance gains using batch calculations on Dell Poweredge 8 core CPU servers belonging to the computer cluster. The last column records similar performance gains on a MacBookPro laptop with I7 Intel 4 core CPU. Columns A, B and D show performance gains obtained by increasing the number o cores, while column C analyzes speed gains by increasing the number of machines (nodes) in the cluster. One can observe that when the maximum number of cores is attained, here 8 for the servers and 4 for the laptop, no additional performance improvement is possible. Performance gains while increasing the number of machines (nodes) can steadily be obtained but, there also, there is a limit when the performance gains become less significant.

Discussions and conclusions

Marketing relies more and more on information technology. From channel choice to personalization and recommendation systems, user-generated content, online reviews, and social influence in social networks (Goes,2014,p.3), Marketing is now considered to be the driver of Big Data technologies, just like accounting was for the databases in the eighties (Albescu & Pugna, 2014). Technology has always been transforming marketing science following a rather systematic and predictable trend. By paraphrasing Rust & Ming-Hui (2014) we would say that, by enabling big customer data access and ubiquitous customer communication, technology pushed marketing science to resemble to a greater degree formerly specialized areas like direct marketing and more recently service marketing. This implies changes in both the topics and methods to be employed and increased emphasis on marketing analytics.

Marketing scientists while having good knowledge in statistics, econometrics, operations research,

seem to have poor knowledge in modern programming and IT. They tend to ignore features that take advantage of the distributed and data-rich context provided by the Internet, cloud computing and HPC. Neglecting factors that enhance usability of models risks to make the latter irrelevant and limit their use. “To develop models that do get used, modelers must pay attention to the IT-infrastructure under which their models will be used.” (Lilien & Rangaswami, 2000, p232)

By trying to demystify BigData approaches our paper invites marketing scientists to pay more attention to technologic evolutions, to become more involved in developing specific analytics and not to leave the battlefield completely to computer scientists. The same invitation goes towards managers who according to an IBM study (2011) of 1,700 CMOs from 19 industries and 64 countries revealed that 71% feel their organizations are unprepared to handle the explosion of big data.

Demythifying BigData approaches and technologies does not mean banalizing them but on the contrary, we insist upon the high importance and ground breaking changes they generate for the human society in general and for marketing science in particular.

This paper presents a hands on approach to BigData in marketing and applies it to rather “big” customer transactions dataset. To our knowledge it is the only explicit application of MapReduce to marketing science problems. Explaining the importance and relative simplicity of this approach can contribute to the adoption of BigData computational techniques among marketing scientists. Although we insist upon the importance of MapReduce and later approaches that hide complexities of parallel computing we extend our study to other HPC technologies that are available to marketing scientists as academic members of universities. The later open new perspectives to leverage opportunities for interdisciplinary collaboration with other units on campus and can help accelerate acquiring BigData expertise.

References

- IBM (2011), *From Stretched to Strengthened – Insights from a Global CMO Study*, <http://www.ibm.com/services/us/cmo/cmstudy2011/downloads.html>, visited on 17/09/2015
- Albescu F., Pugna I.B (2014), Marketing intelligence - the last frontier of business information technologies, *Romanian Journal of Marketing*, 3, 55-68.
- Bracht O. (2013), Five ways to handle Big Data in R, <http://blog.eoda.de/2013/11/27/five-ways-to-handle-big-data-in-r/>
- Calciu M. et Salerno F. (2005) Modélisation prédictive de l'incidence et des montants d'achat en marketing direct: une comparaison a partir de variables RFM., *XXIème Congrès International de l'Association Française du Marketing*, Nancy
- Cass S. (2015), The 2015 Top Ten Programming Languages, *IEEE Spectrum*, <http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages>, posted 20/07/2015, visited on 10/09/2015
- Dean J. & Ghemawat S. (2004), MapReduce: Simplified Data Processing on Large Clusters, *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, December.
- Goes, P., (2014), Big Data and IS Research, *MIS Quarterly*, 38, 3, September, pIII-VIII.
- Lilien G.L & Rangaswamy A. (2000) Modeled to bits: Decision models for the digital, networked economy, *Intern. J. of Research in Marketing*, 17, 227-235

- Manyika, J., Chui, M., Bug- hin, J., Brown, B., Dobbs, R., Roxburgh, C., Byers, A. (2011), *Big Data: The next frontier for innovation, competition, and productivity*. http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation, visited on 01/09/2015.
- Schall M. (2012), The Big Kahuna. *Marketing Research*,24,2, p3
- Wijfels J. (2013), ffbase: statistical functions for large datasets, *The 9th International R User Conference*, Albacete, Castilla-La Mancha, July 10-12
- Rust H.T., Ming-Hui H. (2014), The Service Revolution and the Transformation of Marketing Science, *Marketing Science*, 33,2, 206-221.
- Zaharia M., Chowdhury M., Das T., Dave A., Ma J., McCauley M., Franklin M.J., Shenker S., Stoica I. (2012) Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, *NSDI 2012*, April.

Appendix

Listing 1 - Linear regression by mimicking MapReduce to compare serial to parallel calculations

```
load("~/Documents/Colloques/2016/paris-venise/data/RFM.RData")
y = ifelse(Y$R < 8,1,0)
# returns tXX with tXy on the last column
ftXXtXy = function(X,y) {
  X = as.matrix(cbind(rep(1, nrow(X)),X));cbind(t(X) %*% X, t(X) %*% y)
} # matrix multiplication function to be mapped
parallel.function <- function(i,lX,ly) {
  ftXXtXy(lX[[i]],ly[[i]])
}
library(parallel)
runs = 1;cores = 2:4
cat("          serial  parallel")
for (k in 1:runs) {
  cat(k,"\n")
  for (i in cores) {
    cat("split",i," ")
    sX = split(data.frame(Y), rep(1:i,each = round(nrow(Y) / i + .5,0))[1:nrow(Y)])
    sy = split(y, rep(1:i,each = round(nrow(Y) / i + .5,0))[1:length(y)])
    # serial
    cat(system.time({
      # map the "value" of matrix multiplication to the "key" (index) of each chunk
      tXXtXy = Map(ftXXtXy, sX,sy)
      red = Reduce("+",tXXtXy)
      solve(red[,-5],red[,5])
    })[3]," ")
    # parallel
    cat(system.time({
      results <- mclapply(1:i, parallel.function, sX, sy)
      reduced = Reduce("+",results)
      solve(reduced[,-5],reduced[,5])
    })[3]," \n")
  }
}

X = as.matrix(cbind(rep(1,nrow(Y)),Y))
system.time(solve(t(X) %*% X, t(X) %*% y))
system.time(lm(y ~ ., data = data.frame(cbind(y,Y))))
```

```
...          serial  parallel
split 2  2.011  1.418
split 3  1.839  1.551
split 4  2.171  1.37
```

```
> system.time(solve(t(X) %*% X, t(X) %*% y))
utilisateur      système      écoulé
      0.346       0.089       0.435
```

```
> system.time(lm(y ~ ., data = data.frame(cbind(y,Y))))
utilisateur      système      écoulé
      8.040       0.883       8.935
```

Listing 2 - Logit regression cross-validation to compare multicore to cluster parallel calculations

```
library(parallel)
# We can define a simple cross validation function for glm type objects as follows:
cross_val <- function(i, fm) {
  data <- fm$model
  formula <- formula(fm)
  response <- as.character(terms(fm)[[2]])
  fm <- lm(formula, data = data[-i,])
  newdata <- data[i,]
  # specific glm logit/probit
  predicted <- predict (fm, newdata, type = "response")
  sqrt((predicted - data[i,response]) ^ 2)
}

load("~/Documents/Colloques/2016/paris-venise/data/CharlesBookClub6000.RData")
n = nrow(X)
# glm logit
fm <- glm(Florence ~ R + F + M, family = binomial, data = X)
# implicit multicore parallelism:
system.time({
  quad <- mclapply(1:n, cross_val, fm,
                  mc.cores = 4L, mc.preschedule = TRUE)
})[3]
# explicit cluster parallelism:
cl = makeCluster(2)
clusterExport(cl,c("cross_val","fm"))
system.time({
  quad <- parLapply(cl, 1:2, function(i)
    cross_val(i, fm))
})[3]
stopCluster(cl)
```