Small is beautiful but scalable is better. Scalable Marketing Decision Support Systems for BigData calculations in CRM

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

«Small is beautiful» is often used to praise small, appropriate technologies that are believed to empower people more, and is opposed to "bigger is better". It is the title of a highly successful book on economics that reached millions (1973, 1999). E.F Schumacher, the author who took this phrase from his teacher L. Kohr, was interested not in smallness but in appropriateness of scale as a challenge to the 'too big to fail' showing premonition concerning the bank crisis we witnessed recently. Scalability is the capability of a system to grow or to handle a growing amount of work. It is analogous with economic scalability of a company implying that the underlying business model offers the potential for economic growth within the company. In IT, scalability is influenced by many factors, ranging from syntax details to component abstraction constructs, among which a subtle combination of object-oriented and functional programming is the most relevant. It is probably best embodied in a relatively new computer language called Scala. While object-orientation is immensely successful since the sixties, functional programming enjoyed less “glamour”. By exaggerating a bit, its usefulness became obvious with the advent of the “MapReduce” approach which has revolutionized and democratized BigData solutions and is fundamentally a functional programing concept. For marketing managers who often feel shipwrecked in a see of data - “Data, data everywhere and not a byte to use”, scalable Marketing Decision Support Systems (MDSS) can help transpose the often “small and beautiful” models, that have already been developed or used in marketing science, to new BigData problems. In our paper we address this challenge by introducing scalable MDSS based upon open-source solutions as the recent Apache Spark BigData solution written in Scala that takes MapReduce to the next level by flexibly chaining richer functional expressions as Directed Acyclic Graphs (DAGs) and introducing Resilient Distributed Datasets (RDDs) that make calculations up to 100 times faster. Applications to customer data base predictive modeling will be developed and performance gains will be tested.

Keywords:

marketing decision support systems, scalability, BigData, cloud-computing

Introduction

«Small is beautiful» is often used to praise small, appropriate technologies that are believed to empower people more, and is opposed to "bigger is better". It is the title of a highly successful book on economics that reached millions (1973, 1999). E.F Schumacher, the author who took this phrase from his teacher L. Kohr, was interested not in smallness but in appropriateness of scale as a challenge to the 'too big to fail' showing premonition concerning the global bank crisis we witnessed recently. Scalability is the capability of a system to grow or to handle a growing amount of work. It is analogous with economic scalability of a company implying that the underlying business model offers the potential for economic growth within the company.

MDSS have been first defined by Little (1979, p.11) as "a coordinated collection of *data, models, analytical tools and computing power* by which an organization gathers information and turns it into a basis for action." Classical MDSS are linked to the advent of micro-computers when computer aided decision making became interactive. They were typically based on so called decision calculus models who in order to be used had to be "simple, robust, easy to control, adaptive, as complete as possible and easy to communicate with (Little, 1970, p. 466). Although Little's MDSS definition remains valid today his decision calculus approach defends the "small is beautiful" point of view while fostering marketing models and MDSS usability.

Recently marketing *data* have grown to a point that they often cannot be dealt with by only one computer, it is the case of Big Data. *Computing power,* consisting usually of one personal computer or workstation, can be increased economically by regrouping several commodity computers in a network in order to form a computer cluster or grid that can be managed through cloud flexible scaling. *Models* and *analytical tools* can be adapted using the MapReduce approach and later developments like Apache Spark in order to scale and distribute calculations over computer clusters.

Under such evolutions MDSS while still starting «small and beautiful» should have the capacity to grow in other words, they should become scalable. Scalability affects all the MDSS components listed in the definition above and even the development tools and languages that are used.

For the latter, scalability is influenced by many factors, ranging from syntax details to component abstraction constructs, among which a subtle combination of object-oriented and functional programming is the most relevant. It is probably best embodied in a relatively new computer language called Scala that has been used to build Apache-Spark, the todays most popular BigData processing engine. While object-orientation is immensely successful since the sixties, functional programming enjoyed less "glamour". By exaggerating a bit, its usefulness became obvious with the advent of the "MapReduce" approach which has revolutionized and democratized BigData solutions and is fundamentally a functional programing concept.

For marketing managers who often feel shipwrecked in a see of data - "Data, data everywhere and not a byte to use", scalable Marketing Decision Support Systems (MDSS) can help transpose the often "small and beautiful" models, that have already been developed or used in marketing science, to new BigData problems. In our paper we address this challenge by introducing scalable MDSS. We discuss scalability of the four components of a MDSS as defined by Little (1979): *data, models,*

analytical tools and computing power and insist upon the recent Apache Spark BigData solution written in Scala that takes MapReduce to the next level by flexibly chaining richer functional expressions as Directed Acyclic Graphs (DAGs) and introducing Resilient Distributed Datasets (RDDs) that make calculations up to 100 times faster. Applications to customer data base predictive modeling are developed and performance gains tested.

Scaling the MDSS data component.

What is BigData: Big data in a narrow sense can be defined as data too large to be dealt with by one computer. In a larger sense it has been defined through the 3V model, which has been coined by Laney (2001) as : "high-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision-making". More recently, Gartner (Beyer & Laney, 2012) updated the definition of the 3V information assets as requiring new forms of processing to enable enhanced decision making, insight discovery and process optimization. The 3Vs have been extended in practice to 5V, adding data value and veracity as defining elements.

Palliative strategies for facing BigData on one computer: Dealing with BigData on one computer using available analytics (statistical software like R) can be done by adopting some of the five strategies as suggested by Bracht (2013) : 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. *Bigger hardware* can be a solution when data gets large. 64-bit machines 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, but disk storage can be hundred thousand times slower to access than memory. *Integration of higher performing programming languages* like C++ or Java can speed up statistical software performance. *Alternative interpreters* can also be used if they exist for a given statistical software.

Although a discussion on MDSS data components can be extended to data format, database, no-sql DB or even datawarhouse aspects we concentrate here on the network data storage approaches that facilitate BigData calculations: Virtual volumes, NFS, HDFS and RDDs.

Virtual volumes on a cloud are persistent storage devices that may be attached and detached from computer instances, like an external hard drive. They do not provide shared storage in the way a network file system (NFS) does. We used a virtual volume to store the two big data files we have used for this research: a 9Gb file containing 344 million customer transactions and a 55 Gb file containing 82.68 million Amazon customer reviews (curtesy McAuley, Padey &Leskovec, 2015).

To share these files with the other computer instances in a cluster we configured a Network File System (NFS) in order to allow them to view/change those files as if they were stored on the local machine.

Hadoop Distributed File System (HDFS) which is the preferred way to store BigData for

calculations can also do that, but it is additionally distributed, fault-tolerant and scalable. It provides very high aggregate bandwidth across the cluster. Fault tolerance in this case is achieved through data replication (3 times). Fault tolerance is necessary in distributed calculations, because the probability of a fault in a cluster of computers increases exponentially with the size of that cluster.

Resilient Distributed Datasets (RDDs) besides being the main idea behind Spark the BigData engine that increased calculation speeds up to 100 times compared to previous MapReduce solutions, achieve fault tolerance through a notion of lineage (see DAG further): if a partition of an RDD is lost, the RDD has enough information to rebuild just that partition. This removes the need for replication to achieve fault tolerance. Besides resilience which is defined as the ability (of the network) to provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation, RDDs are ".. parallel data structures that let users explicitly persist intermediate results in memory, control their partitioning to optimize data placement, and manipulate them using a rich set of operators" (Zaharia et al., 2012)

Scaling the MDSS computing power component

Today's mainstream computer hardware is relatively cheap and almost infinitely replicable. It is much cheaper to purchase 8 off-the-shelf, "commodity" servers with eight processing cores and 128 GB of RAM each than to acquire a single system with 64 processors and a terabyte of RAM. While there exist other alternatives to grow computing power for analyzing very large datasets, the most successful, flexible and economic way is by distributed computing usually accessed though a cloud. Cloud computing provides the tools and technologies to build data/compute intensive parallel applications with much more affordable prices compared to traditional parallel computing techniques.(Hamdaqua & Tahvildari, 2012)

The American National Institute of Standards and Technology (NIST) defines c*loud computing* "as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models ." (Mell & Grance, 2011, p.2) .

Fig.1 - NIST visual model of cloud computing definition.

Essential Characteristics	Broad Network Access	Rapid Elasticity	Measured Service	On-Demand Self-Service	Resource Pooling
Service Models	SaaS (Software as a Service)		PaaS (Platform as a Service)		IaaS (Infrastructure as a Service)
Deployment Models	Public	Private		Hybrid	Community

Adapted from Hamdaqua & Tahvildari (2012)

The main enabling technology for cloud computing is virtualization. Virtualization software separates a physical computing device (bare metal environment) into one or more "virtual" devices, each of which can be easily used and managed to perform computing tasks. With operating system level virtualization essentially creating a scalable system of multiple independent computing

devices, idle computing resources can be allocated and used more efficiently. *Virtualization* provides the agility required to speed up IT operations, and reduces cost by increasing infrastructure utilization. *Autonomic computing* automates the process through which the user can provision resources on-demand. By minimizing user involvement, automation speeds up the process, reduces labor costs and reduces the possibility of human errors. (Hamdaqua & Tahvildari, 2012).

Implementing the scalable MDSS computing power component

Our MDSS computing power component is hosted on the cloud of Lille University[1] that is based upon OpenStack, a free and open-source software platform for cloud computing, mostly deployed as an infrastructure-as-a-service (IaaS).

Prepare the local (client) computer to use the cloud: Although OpenStack has a quite powerful web-interface which allows to launch virtual computer instances of various linux system versions and flavors from available images these images usually don't have big-data calculation software installed. Therefore the main command-line open-stack clients need to be installed (keystone, nova, glance, cinder, neutron) in order to automate install operations by preparing shell commands. A configuration file (here cloud-hpc-lille.rc) to create the client environment consisting of username and password (also tenant name and the server's authentication url) allows the marketing scientist to remotely control the scalable MDSS infrastructure.

Controlling the remote scalable MDSS infrastructure: Before launching several virtual computer instances that will form a cluster it is necessary to generate the ssh key[2] on the local computer and register it with the cloud[3] in order to be able to remotely access the cloud administration server and control the MDSS infrastructure.

Controlling the MDSS infrastructure means launching virtual computer instances by choosing the operating system among several linux distribution images (in our case Ubuntu 14.04.2 - 64 bit) and a given flavor (in our case 8-CPU-12GB-RAM) as well as the network on which these virtual computers will form a cluster.

Listing 1 - Creating the cluster infrastructure on the cloud with OpenStack

```
openstack server create --flavor $OS_FLAVOR --image $OS_IMAGE --nic net-
id=$OS_NETID --key-name $OS_KEYNAME --user-data init-spark-master.sh spark-
master # Create Spark master VM
for i in {1..7}
do
openstack server create --flavor $OS_FLAVOR --image $OS_IMAGE --nic net-
id=$OS_NETID --key-name $OS_KEYNAME --user-data init-spark-worker.sh spark-
worker$i # Create first Spark workers
done
```

The launched instances can then be interactively or programmatically observed, stopped, started or even deleted.

The size of the cluster can be adapted within the limits of the quota previously attributed by the

1 The cloud consists of 336 cores, 2.1 To RAM and 215 To de storage and a total power of de 6,5 Tflops
2 ssh-keygen -t rsa -f ${HOME}/.ssh/cloud-hpc-lille
3 nova keypair-add --pub-key=${HOME}/.ssh/cloud-hpc-lille.pub mcmac2key

cloud administrator[4]. Our quota consisted of 8 instances with 8 cores each (meaning a total of 64 cores),12 Gb RAM (meaning 96Gb for the cluster) and 25 Gb disk storage (meaning potentially 200 Gb for the HDFS file system). For the BigData calculations engine (here Apache-Spark) one computer instance will be given the role of a master and the others the role of workers (slaves).

Another aspect of Infrastructure as a Service (IaaS) is controlling network infrastructure by associating public IP addresses to some computers in the clusters and ensuring their firewall protection through association to security groups. Public IP addresses allow among others connecting to and controlling the instances and viewing on the web browser the calculation stages of the BigData cluster engines that will be installed on each computer.

Scaling the MDSS model and analytic components for Big Data

Traditional data analysis algorithms and techniques are not well suited for Big data. They do not scale properly as the data size increases. Therefore, scalable methodologies and frameworks have been developed that can split data that are too big for one computer over several computers and organize calculations by hiding complexities of parallel computing while exposing some functional programming facilities.

Functional programming is a paradigm that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. It is a declarative programming paradigm, which means programming is done with expressions or declarations instead of statements as in opposite *imperative programming*. Some of attributes of functional programming have been used in solutions that have democratized BigData calculations. These are the map and reduce higher order functions that have inspired the well known MapReduce approach (Dean & Gwenmawat, 2004, 2008) to BigData.

MapReduce with applications to CRM calculations

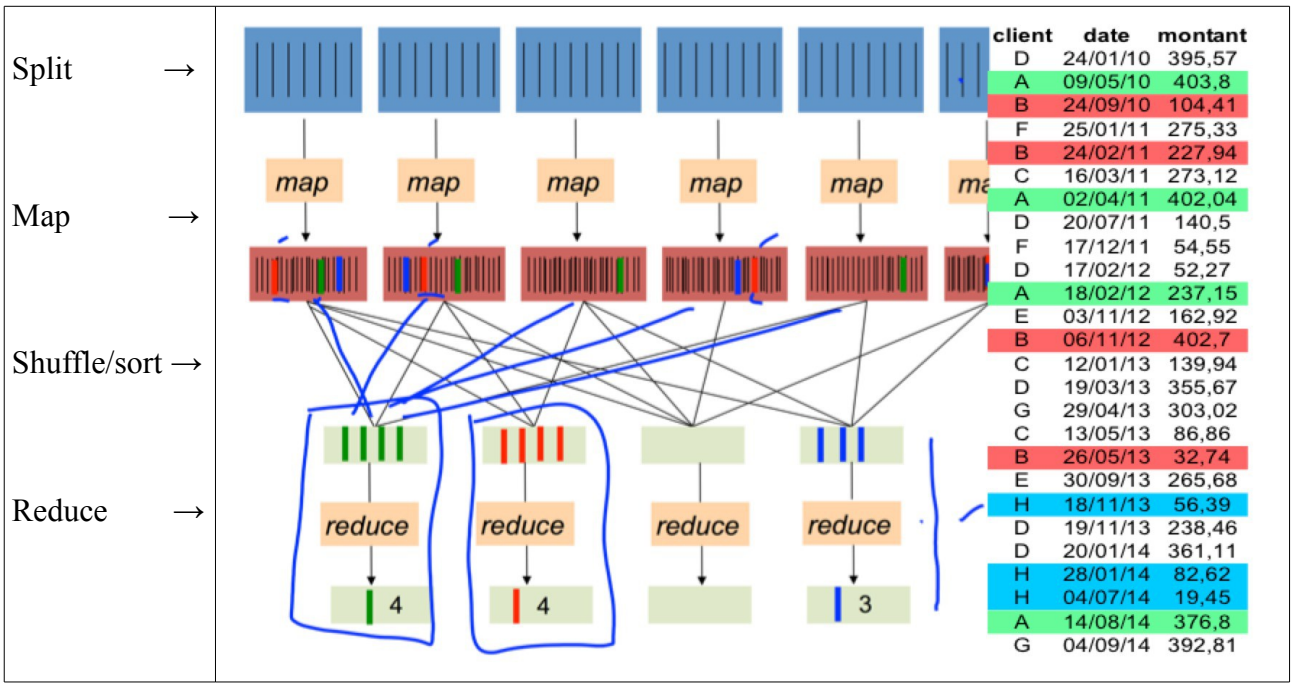
What is MapReduce: MapReduce is a high level programming model and an associated implementation for large scale parallel data processing. It has the merit to hide all complexities of parallel computing on distributed servers from users and to have contributed massively to democratize BigData processing. The name MapReduce originally referred to the proprietary Google technology (Dean and Ghemawat, 2004), but has since become a generic trademark. It's most popular implementation is part of 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.

MapReduce is based on the observation that most computations can be expressed in terms of a Map() procedure that performs filtering and sorting (such as sorting customer transactions by name, date or amount into queues, one queue for each name) and of a Reduce() procedure that performs an aggregating operations (such as counting the number of transactions in each queue, yielding customer purchase **F**requency, or retaining the maximum date, yielding customer **R**ecency or summing transactions amount, yielding customer **M**onetary amount). Figure 2 illustrates the main stages of a MapReduce process applied to customer base Recency (R), Frequency (F) and Monetary

4 nova show-quota

(M) calculations.

Figure 2 - MapReduce example: RFM calculation on a customer base



While marketing scientists may concentrate on the map and reduce function, the key contributions of the MapReduce framework are hidden in the split and shuffle/sort stages that achieve scalability and fault-tolerance for a variety of applications by optimizing the execution engine. (Zaharia et al., 2008).

In the example above each customer transaction, as can be seen on the right hand side of Figure 2, 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.

The MapReduce process (see Figure 2) reads the input file and splits 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 each chunk to form a queue per customer. The system then takes the output from each map program and merges (shuffle/sort) the results for input into the reduce program, which calculates per customer, the length (or sum of values) of the queue for Frequency, the maximum value for Recency and the sum or the average value for the purchase amount or Monetary.

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, Principal Components 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. A special case of matrix multiplication that is central to the above mentioned data analysis methods is the multiplication of the transposed matrix with itself. This matrix multiplication has a rather straightforward solution with MapReduce that exploits the fact that 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.

Apache-Spark an evolution based upon MapReduce

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. MapReduce is not suitable for all problems, new programming models and frameworks are still being created that build upon these ideas.

The most popular is Apache-Spark which implements DAGs (Directed Acyclic Graph) and RDDs (Resilient Distributed Dataset). DAG is 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 stages that can form a lineage or a tree structure and is therefore more flexible due to more high level functions like map, filter, union etc. Also DAG execution is faster due to intermediate results not being written to disk. RDDs, have been mentioned earlier when we discussed scalable MDSS data components and are best explained by the initiator Zaharia & al. (2012) and Zaharia (2014) in a disertation submitted in partial satisfaction for the PhD degree from Berkley University. Spark takes MapReduce to the next level with less 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.

Spark is also very concise in terms of lines of code. Its first published version was only 14000 lines of code written in Scala, while Hadoop's MapReduce that was written in Java was almost ten times bigger (Zaharia, 2014). While Spark has replaced MapReduce which was so closely associated to Hadoop, the two solutions are complementary. Hadoop remains essentially a distributed data infrastructure and a format for storing data, that can be processed by Spark or other projects.

Scalability of MDSS development tools

MDSS can and have been developed using various tools like spread-sheets, database systems and specialized or general purpose computer languages. In IT, scalability is influenced by many factors, ranging from syntax details to component abstraction constructs, among which a subtle combination of object-oriented and functional programming is the most relevant. It is probably best embodied in a relatively new computer language called Scala. "..in Scala a function value is an object. Function types are classes that can be inherited by subclasses. This might seem nothing more than an academic nicety, but it has deep consequences for scalability". (Odersky & al, 2011, p.55). While object-orientation is immensely successful since the sixties, functional programming enjoyed less "glamour". By exaggerating a bit, its usefulness became obvious with the advent of the

“MapReduce” approach which has revolutionized and democratized BigData solutions and is fundamentally a functional programing concept.

It is a real dilemma wether to choose as MDSS development tools specialized statistical and mathematical languages like R or Matlab or general purpose languages like Scala, Java or Python.

The R statistical system progressively becomes the preferred tool for a majority of data analysts and for an increasing number of marketing scientists. R is both functional and object oriented. 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. For applications of R in BigData CRM calculations one could refer to Calciu, Moulins, Salerno (2016)

While R scalability is implicit in the Scala language it is explicit, that is also what its name stands for. For more details about scalability, elegance and conciseness of Scala on could read Odersky et al. (2011). Also in order to better understand what recommends Scala as a platform for statistical computing and data science one could refer to D. Wilkinson’s (2013) research blog articles.

MDSS for CRM calculations using Spark with Scala

We used Scala with Spark to implement our MDSS solutions. Spark with its over 80 high-level operators is much more expressive when implementing analytics than MapReduce where problems had to be forced to fit within the narrow constraints of the map and reduce stages. To illustrate let’s take the classic Word Count example. Written in Java for MapReduce it has around 100 lines of code and is rather unintuitive, whereas in Spark (and Scala) it takes not more than 4 lines as in Listing 2.

Listing 2 - Word Count program using MapReduce coded in Scala on Spark

```
1. sparkContext.textFile("hdfs://...")
2.  .flatMap(line => line.split(" "))
3.  .map(word => (word, 1)).reduceByKey(_ + _)
4.  .saveAsTextFile("hdfs://…")
```

After reading the text file in a RDD the second command splits each line of text in an array of words using spaces as a split separator. The third line maps a key/value pair for each word where the key is the word and the value is 1 (one). The reduceByKey higher order function sums all values (ones) for each word (key), obtaining the word frequency.

Computing RFM variables for CRM modeling

Computing customer Frequency, Recency and Monetary values is a very similar exercise.

Listing 3 - Computing RFM variables on Apache Spark using Scala

```
1. val textFile = sc.textFile("/media/storage1/sX")
2. case class Transaction(custid: BigInt, date: java.util.Date, M: Double)
3. val dateformat = new java.text.SimpleDateFormat("dd/MM/yyyy")
4. val transactions = textFile.filter(!_.contains("client")).map(_.split("
   ")).map(c => Transaction(BigInt(c(0)),dateformat.parse(c(1)),c(2).toDouble))
5. val custF = transactions.map(t => (t.custid, 1)).reduceByKey(_ + _)
```

```
6. val custM = transactions.map(t => (t.custid, t.M)).reduceByKey(_ + _)
7. val custR = transactions.map(t => (t.custid, t.date)).reduceByKey((a,b) => if
   (a after b) a else b)
```

In this case (see Listing 3) we take a text file as the one that can be seen in figure 2, after reading the file into an imutable RDD called textFile, we apply a series (lineage, DAG) of transformations in line 4 to filter out the header line detected with the word “date”, then split the text in each line using the “ “ separator and finally transform each of the three text elements into a Transaction object whose class has been defined in line 2. The result is a transactions RDD on which we apply map and reduce higher order function as in the Wordcount listing (see listing 2) they are identical for customer Frequency (line 5) and Monetary (line 6) and for the customer Recency (line 7) as the value in the mapped key/value pair is the date in the reduce phase we apply an anonymous function that takes the maximum date and not the sum as before.

Measuring consumer sentiment using Spark’s mathematical library

Measuring customer sentiment towards products and/or brands is a typical marketing application that needs BigData calculation. We have used the Amazon customer reviews dataset mentioned earlier that contains 82.68 million reviews after deduplication (142.8 million reviews originally) spanning May 1996 - July 2014. The two first reviews separated by square brackets in json (JavaScript Object Notation) format are given in Listing 4.

Listing 4 - The First two records in the Amazon Reviews Dataset

```
{"reviewerID": "A00000262KYZUE4J55XGL", "asin": "B003UYU16G",
"reviewerName": "Steven N Elich", "helpful": [0, 0], "reviewText": "It
is and does exactly what the description said it would be and would do.
Couldn't be happier with it.", "overall": 5.0, "summary": "Does what
it's supposed to do", "unixReviewTime": 1353456000, "reviewTime": "11
21, 2012"}
{"reviewerID": "A000008615DZQRRI946FO", "asin": "B005FYPK9C",
"reviewerName": "mj waldon", "helpful": [0, 0], "reviewText": "I was
sketchy at first about these but once you wear them for a couple hours
they break in they fit good on my board an have little wear from skating
in them. They are a little heavy but won't get eaten up as bad by your
grip tape like poser dc shoes.", "overall": 5.0, "summary": "great buy",
"unixReviewTime": 1357603200, "reviewTime": "01 8, 2013"}
```

Measuring consumer sentiment is a typical document classification task, the input to the machine learning algorithm (both during learning and classification) is the customers’ ratings and reviews text. The calculations that are listed in the Appendix, are organized in a pipeline that chains several operations: tokenizing, hashing with term frequency and a regression (see line 10). The tokenizer extracts (and counts) the individual words (line 7). For the review text a bag of words (BOW) representation is constructed. Each distinct word (token) in the training set defines a feature (independent variable) of each of the reviews in both the training and test sets. HashingTF (line 8) is a transformer which takes BOWs and converts them into fixed-length feature vectors. After this stage a “featureized” short review text would look like this:

```
Array([0,Hi I heard about Spark, WrappedArray(hi, i, heard, about, spark),(20,
```

```
[0,5,9,17],[1.0,1.0,1.0,2.0])])
```
It is an array containing a Vector with 4 positions. The first is the identifier of the customer, the second the review text, the third a Wrapped Array with the tokenized words and the last a sparse matrix with 20 columns corresponding to the number of distinct words used in a small set of mock reviews and 2 rows. The first row indicates the position of the word in the total word list and the second shows how frequently the word was used.

Lasso regression uses a form of Regularized Least Squares that like Ridge regression is suited when the number of independent variables is big and has the advantage over the latter to automatically select more relevant features and discards the others. That's why it has been chosen in order to regress words used in reviews against ratings given by customers. If a linear regression model is trained with the elastic net parameter α set to 1, it is equivalent to a Lasso model (line 9).

After defining the pipeline that chains calculations, an initial parameter grid is prepared (lines 12-14) and also an evaluator that uses R2 as a goodness of fit metric (line 18).

Finally the cross-validation procedure combines the pipeline as an estimator, the evaluator and the parameter grid (line 20) while real calculations occur much later when cross-validation is launched to fit over the training data (line 24).

Once the model has been estimated over the reviews training-set its performance can be evaluated over the test-set (line 28) with the R2 score (line 29) and rating predictions (consumer sentiment ratings) can be given by using the calibrated model with customer reviews from the test set (line 32).

Conclusion and discussions

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). Important market transformations have been generated by leading e-commerce enterprises such Amazon and eBay through their innovative and highly scalable e-commerce platforms and recommender systems (Bello-Orgaza & al., 2016). Amazon is also a forerunner in cloud technology with its Amazon Web Services (AWS) and especially its Elastic Cloud Computing Services (EC2). The cloud becomes more and more the place where BigData calculations are done. Under these circumstances MDSS and marketing models can no longer remain "small and beautiful". They must be capable to grow, that is become scalable.

To our knowledge this is the first academic attempt to define and implement scalable MDSS on a cloud for marketing BigData calculations. It is also probably the first to use the new Apache-Spark cluster computing system that has replaced the famous MapReduce approach which has been initiated by Google and has dominated BigData calculations on computer clusters over the last decade especially through its open-source implementation on the Apache-Hadoop engine. As BigData calculations tend to last longer than the model building process and are in most cases launched in batch mode, we didn't address MDSS user interface aspects. Interactive user interfaces can be easily implemented using modern web client and server technologies.

We have shown that Little's (1979) seminal definition of MDSS, as a collection of *data, models,*

analytical tools and computing power, witnessed sufficient premonition to remain valid in the new milenium and to allow for MDSS scalability accommodation. We demonstrate how this can be done using most recent open-source solutions. For the scalability of the data component we have focused on those solutions that directly facilitate distributed calculations over computer clusters such as virtual volume storage, NFS, HDFS and RDDs, the latter two having different approaches to fault-tolerance which is so important in cluster computing. For scalability of computing power we have privileged Infrastructure as a Service (IaaS) cloud solutions. For scaling models and analytics we have explained how the MapReduce approach by hiding complexities of parallel computing has largely contributed to democratize BigData calculations and also why MapReduce is no longer the preferred solution and is being replaced by Apache-Spark. More fundamentally we have discussed scalability of MDSS development tools and how to detect it and also the dilemma of choosing among tools specialized for analytics or general purpose tools.

If the necessity to use R for modeling and analytics in marketing has become obvious and has been extensively discussed by us in a previous paper, we also evoke the possibility to adopt Scala for its "scalability", elegance, conciseness and the capability of its ecosystem to deal with BigData.

Our scalable MDSS implementations address CRM BigData calculation issues and rely exclusively on open-source software.

References:

Bello-Orgaza, G., Jungb J.J, Camachoa D. (2016) Social big data: Recent achievements and new challenges, *Information Fusion,* 28, 45–59

Beyer, M.A., Laney D. (2012) *The Importance of 'Big Data': A Definition*, Gartner, Stamford, CT

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/

Bradley, J. (2016) *Apach Spark Mlib: From Quickstart to Scikit-Learn*, http://go.databricks.com/spark-mllib-from-quick-start-to-scikit-learn, (accessed November 2016)

Calciu, M., Moulins J-L., Salerno F. (2016) Big data and open-source computation solutions, opportunities and challenges for marketing scientists. Applications to customer base predictive modeling using RFM variables, *15th International Marketing Trends Conference*, Venice, January.

Dean, J., Ghemawat S. (2004) Mapreduce: simplified data processing on large clusters, in: *Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation*, OSDI'04, USENIX Association, 2004.

Dean J., Ghemawat S. (2008) Mapreduce: simplified data processing on large clusters, *Commun. ACM,* 51(1) (2008) 107–113,doi:10.1145/1327452.1327492.

Hamdaqa, M., Tahvildari L. (2012) Cloud Computing Uncovered: A Research Landscape, *Advances in Computers*, 86, 41, 43-84, http://dx.doi.org/10.1016/B978-0-12-396535-6.00002-8

Laney D. (2001) *3D Data Management: Controlling Data Volume, Velocity, and Variety, Technical Report*, U R L https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf (accessed September 2016).

Lin J., Ryaboy D. (2012) Scaling the Big Data Mining Infrastructure. The Twitter Experience, *SIGKDD Explorations*, 14, 2, 5-19

Little J.D.C (1979) Models and Managers: The Concept of a Decision Calculus, *Management Science*, 16, 8 (April), 467-485.

Little J.D.C (1979) Decision Support Systems for Marketing Managers, *Journal of Marketing*, 3 (Summer), 9-25

McAuley, J., Pandey, R. & Leskovec J (2015) Inferring networks of substitutable and complementary products, *KDD '15 Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 785-794

Mell, P., Grance T. (2009) The NIST definition of cloud computing, *Recommendations of the National Institute of Standards and Technology Special Publication 800-145*, National Institute of Standards and Technology

Odersky, M., Spoon L., Venners B. (2011) *Programming in Scala, 2nd Edition: A comprehensive step-by-step guide*, 2 edition (January 4, 2011), Artima Inc

Schumacher E.F. (1973) Small Is Beautiful: A Study of Economics As If People Mattered,Blond & Briggs

Wilkinson D. (2013) Scala as a platform for statistical computing and data science, URL: https://darrenjw.wordpress.com/2013/12/23/scala-as-a-platform-for-statistical-computing-and-data-science/,(accessed September 2016)

M.Zaharia, A.Konwinski, A.D.Joseph, R.Katz, I.Stoica, (2008) Improving mapreduce performance in heterogeneous environments, in: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI'08, USENIX Association, Berkeley, CA, USA, 2008, pp.29–42.http://dl.acm.org/citation.cfm?id=1855741.1855744.

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.

Zaharia M. (2014) An Architecture for Fast and General Data Processing on Large Clusters, University of California at Berkeley, *Technical Report No. UCB/EECS-2014-12*.

Appendix

Listing 5 - Measuring customer sentiment on the Amazon Reviews Dataset*

```
# Load dataset from Parquet format, and cache it
trainFilepath = "/media/storage1/review-training"
testFilepath = "/media/storage1/review-test"
data = sqlContext.read.format("parquet").load(trainFilepath).cache()

# Define a pipeline combining text feature extractors + linear regression
tokenizer = Tokenizer(inputCol="review", outputCol="words")
hashingTF = HashingTF(inputCol="words", outputCol="features")
lasso = LinearRegression(labelCol="rating", elasticNetParam=1.0, maxIter=20)
   pipeline = Pipeline(stages=[tokenizer, hashingTF, lasso])

   paramGrid = ParamGridBuilder()\
    .addGrid(lasso.regParam, [0.005, 0.01, 0.05])\
    .build()


   # Define evaluation metric
   evaluator = RegressionEvaluator(labelCol="rating", metricName="r2")

   cv = CrossValidator(estimator=pipeline, evaluator=evaluator,
  estimatorParamMaps=paramGrid)


   # Run everything!
   cvModel = cv.fit(data)

   Evaluate on test data:

   test = sqlContext.read.format("parquet").load("/media/storage1/review-
  test")
   r2 = evaluator.evaluate(cvModel.transform(test))
   print "Test data R^2 score: %g" % r2

   sparkPredictions = cvModel.transform(test)
   sparkPredictions.write.format("json").mode("overwrite").save(predictionsFi
  lePath)
```

** The listing is adapted from Bradley (2016)*