A MAPREDUCE SOLUTION FOR HANDLING LARGE DATA EFFICIENTLY

M.Sc. K. Al-Barznji PhD Student, Assoc. Prof. Dr. A. Atanassov
Department of Computer Science, University of Chemical Technology and Metallurgy, Sofia, Bulgaria

kamal.barznji@raparinuni.org , nasa@uctm.edu

Abstract: *Big data is large volume, heterogeneous, distributed data. Big data applications where data collection has grown continuously, it is expensive to manage, capture or extract and process data using existing software tools. With increasing size of data in data warehouse it is expensive to perform data analysis. In recent years, numbers of computation and data intensive scientific data analyses are established. To perform the large scale data mining analyses so as to meet the scalability and performance requirements of big data, several efficient parallel and concurrent algorithms got applied. For data processing, Big data processing framework relay on cluster computers and parallel execution framework provided by MapReduce. MapReduce is a parallel programming model and an associated implementation for processing and generating large data sets. In this paper, we are going to work around MapReduce, use a MapReduce solution for handling large data efficiently, its advantages, disadvantages and how it can be used in integration with other technology.*

Keywords: DATA MINING; BIG DATA; CLUSTERING; PARALLEL PROCESSING; HADOOP; HDFS; MAPREDUCE;

1. Introduction

Data Mining is analyzing the data from different perspectives and summarizing it into useful information that can be used for business solutions and predicting the future trends. Mining the information helps organizations to make knowledge driven decisions. Data mining (DM), also called Knowledge Discovery in Databases (KDD), is the process of searching large volumes of data automatically for patterns such as association rules. It applies many computational techniques from statistics, information retrieval, machine learning and pattern 1recognition. Data mining extract only required patterns from the database in a short time span. Based on the type of patterns to be mined, data mining tasks can be classified into summarization, classification, clustering, association and trends analysis [1]. Among several techniques in data mining, clustering is the main considerable point which is used to retrieve the essential knowledge from the very huge collection of data. Clustering can handle both the complicated and very large datasets which can be of diverse data types. To classify the large data sets clustering is the vital solution [2]. Big Data is a new term used to identify the data sets that are of large size and have grater complexity [3]. Big data is defined as large amount of data which requires new technologies and architectures to make possible to extract value from it by capturing and analysis process [4].

Over the last few years, MapReduce has emerged as the most popular computing paradigm for parallel, batch style and analysis of large amounts of data. Many areas where massive data analysis is required, MapReduce are used. There are evolving numbers of applications that handle big data, but to handle such huge collection of data is a very challenging problem today. Here, we got the MapReduce or its open source equivalent Hadoop which is a powerful tool for building such applications [5]. MapReduce divides the computational flow into two main phases: **Map** and **Reduce**. By simply designing Map and Reduce functions, developers are able to implement parallel algorithms that can be executed across the cluster [6]. The steps involved in working of MapReduce can be shown in as **figure 1** [7]:

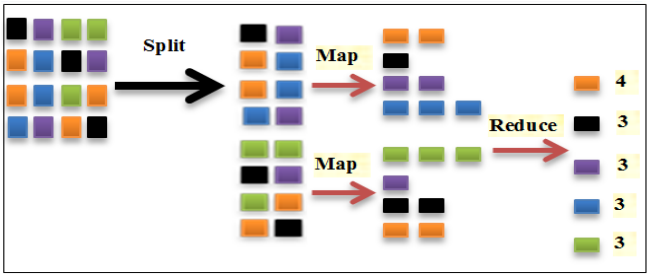

Fig.1. *Steps in MapReduce to process the database*

Data intensive processing is fast and currently becoming a necessity to handle the large databases efficiently. It is required to design algorithms that must be capable of scaling to real world datasets. MapReduce is a programming model which is inspired by functional programming which allows expressing distributed computations on massive amounts of data. It is designed for large scale data processing as it allows running on clusters of commodity hardware [8]. MapReduce is a powerful parallel programming technique for distributed processing of vast amount of data on clusters [9].

2. Background

In this section, we review the Apache Hadoop Fundamentals (HDFS and MapReduce) Explained with a Diagram**:**

2.1. Hadoop Framework

Apache Hadoop is an open source Java framework for processing and querying vast amounts of data on large clusters of commodity hardware. With a significant technology investment by Yahoo!, Apache Hadoop has become an enterprise ready cloud computing technology. It is becoming the industry defacto framework for Big Data processing. Hadoop is impact can be boiled down to four salient characteristics. Hadoop enables scalable, costeffective, flexible, fault tolerant solutions [10]. Hadoop was derived from Google's MapReduce and Google File System (GFS) [11]. Hadoop has **two** primary components, namely, **HDFS** and **MapReduce** programming framework. The most significant feature of Hadoop is that HDFS and MapReduce are closely related to each other. Therefore, the storage system is not physically separated from the processing system [12]. The following **figure 2** shows the Hadoop framework main components [13].

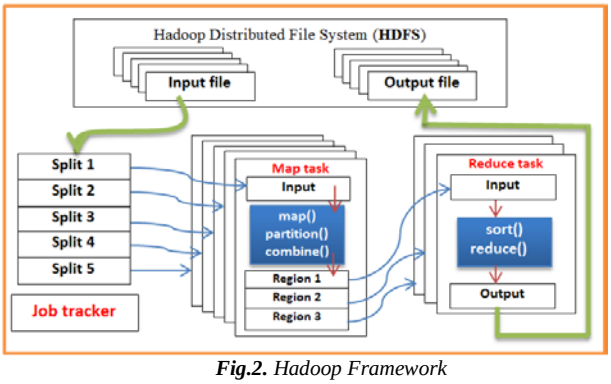

Fig.2. *Hadoop Framework*

2.1.1. HDFS (Hadoop Distributed File System)

HDFS stands for Hadoop Distributed File System, it is the

storage system used by Hadoop. The following is a high level architecture that explains how HDFS works(see **figure 3)** [21].

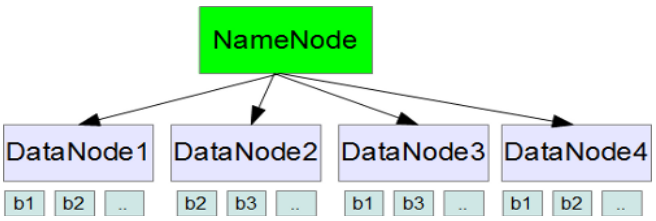

***Fig.3**. HDFS Diagram.*

- In the above diagram, there is one NameNode, and multiple DataNodes(servers). b1, b2, indicates data blocks.
- When you dump a file (or data) into the HDFS, it stores them in blocks on the various nodes in the hadoop cluster. HDFS creates several replications of the data blocks and distributes them accordingly in the cluster in way that will be reliable and can be retrieved faster. A typical HDFS block size is 128MB. Each and every data block is replicated to multiple nodes across the cluster.
- Hadoop will internally make sure that any node failure will never results in a data loss.
- There will be one NameNode that manages the file system metadata.
- There will be multiple DataNodes (These are the real cheap commodity servers) that will store the data blocks.
- When you execute a query from a client, it will reach out to the NameNode to get the file metadata information, and then it will reach out to the DataNodes to get the real data blocks
- Hadoop provides a command line interface for administrators to work on HDFS.
- The NameNode comes with an in-built web server from where you can browse the HDFS filesystem and view some basic cluster statistics.

2.1.2. MapReduce Framework

MapReduce is a software framework introduced by Google in 2004 to support distributed computing on large data sets on clusters of computers. MapReduce is a programming model for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs and a reduce function that merges all intermediate values associated with the same intermediate key [11]. MapReduce is designed to be used by programmers, rather than business users. It is a programming model, not a programming language. It has gained popularity for its easiness, efficiency and ability to control Big Data in a timely manner [7]. Google proposed the MapReduce programming framework based on functional programming. The framework divides the work into independent tasks and parallelizes the computation flow across large scale clusters of machines, taking care of communications among them and possible failures, and efficiently handling network bandwidth and disk usage [6]. The following is a high-level architecture that explains how **MapReduce** works (see **figure 4**) [21].

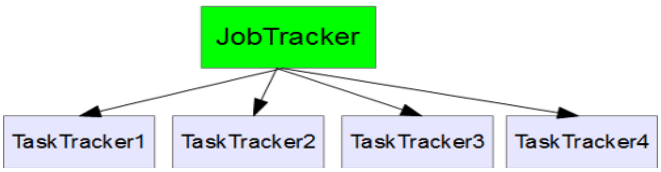

***Fig.4**. MapReduce Diagram*

- MapReduce is a parallel programming model that is used to retrieve the data from the Hadoop cluster
- In this model, the library handles lot of messy details that programmers doesn't need to worry about. For example, the library takes care of parallelization, fault tolerance, data distribution, load balancing, etc.
- This splits the tasks and executes on the various nodes parallely, thus speeding up the computation and retriving required data from a huge dataset in a fast manner.
- This provides a clear abstraction for programmers. They have to just implement two functions: map and reduce.
- The data are fed into the map function as key value pairs to produce intermediate key/value pairs
- Once the mapping is done, all the intermediate results from various nodes are reduced to create the final output
- JobTracker keeps track of all the MapReduces jobs that are running on various nodes. This schedules the jobs, keeps track of all the map and reduce jobs running across the nodes. If any one of those jobs fails, it reallocates the job to another node, etc. In simple terms, JobTracker is responsible for making sure that the query on a huge dataset runs successfully and the data is returned to the client in a reliable manner.
- TaskTracker performs the map and reduce tasks that are assigned by the JobTracker. TaskTracker also constantly sends a hearbeat message to JobTracker, which helps JobTracker to decide whether to delegate a new task to this particular node or not.

3. Mapreduce Architecture

In most computation related to high data volumes, it is observed that two main phases are commonly used in most data processing components this is shown in below **figure 5.** Map Reduce created an abstraction phases of Map Reduce model called **mappers** and **reducers**. When it comes to processing large data sets, for each logical record in the input data it is often required to use a mapping function to create intermediate key value pairs. Then another phase called reduce to be applied to the data that shares the same key, to derive the combined data appropriately [14].

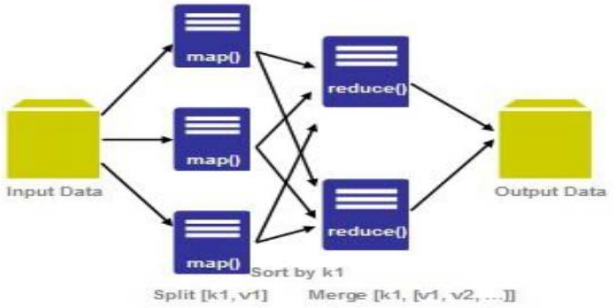

***Fig.5.** MapReduce Architecture*

Mapper

The mapper is applied to every input key-value pair to generate an arbitrary number of intermediate key-value pairs. The standard representation of this is as follows:

map(inKey,inValue)→list(intermediateKey, intermediateValue).

The purpose of the map phase is to organize the data in preparation for the processing done in the reduce phase. The input to the map function is in the form of key-value pairs, even though the input to a MapReduce program is a file or file(s). By default, the value is a data record and the key is generally the offset of the data record from the beginning of the data file. The output consists of a collection of key-value pairs which are input for the reduce function. The content of the key-value pairs depends on the specific implementation.

For example, a common initial program implemented in MapReduce is to count words in a file. The input to the mapper is each line of the file, while the output from each mapper is a set of key-value pairs where one word is the key and the number 1 is the value.

map: (k1 , v1) → [(k2 , v2)]

The file_name and the file_content which is denoted by k1 and v1. So, with in the map function user may emit the any arbitrary key/value pair as denoted in the list [k2, v2]. To optimize the processing capacity of the map phase, MapReduce can run several identical mappers in parallel. Since every mapper is the same, they produce the same result as running one map function.

Reducer

The reducer is applied to all values associated with the same intermediate key to generate output key-value pairs.

reduce(intermediateKey,list(intermediateValue))--> list(outKey, outValue)

Each reduce function processes the intermediate values for a particular key generated by the map function and generates the output. Essentially there exists a one-one mapping between keys and reducers. Several reducers can run in parallel, since they are independent of one another. The number of reducers is decided by the user. By default, the number of reducers is 1. Since we have an intermediate group by operation, the input to the reducer function is a key value pair where the key- k2 is the one which is emitted from mapper and a list of values [v2] with shares the same key.

reduce: (k2 , [v2]) → [(k3 , v3)] .

4. Programming Model

MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. The nature of this programming model and how it can be used to write programs which run in the Hadoop environment is explained by this model. Hadoop is an open source implementation of this environment [15].

The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: map and reduce. Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the reduce function. The reduce function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges these values together to form a possibly smaller set of values. Typically just zero or one output value is produced per reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory. MapReduce [22] requires that the operations performed at the reduce task to be both associative and commutative. This two stage processing structure is illustrated in **Figure 6:**

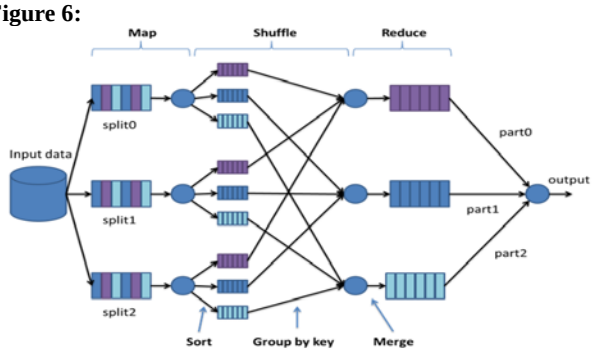

***Fig.6**. MapReduce simplified flowchart*

Map and Reduce are two functions. The main job of these two functions are sorting and filtering input data. During Map phase data is distributed to mapper machines and by parallel processing the subset it produces <key, value>pairs for each record. Next shuffle phase is used to repartition and sorting that pair within each partition. So the value corresponding same key grouped into {v1, v2,….} values. Last during Reduce phase reducer machine process subset <key, {v1, v2,}>pairs parallel in the final result is written to distributed file system [15]. A Hadoop MapReduce program also has a component called the Driver. The driver is responsible for initializing the job with its configuration details, specifying the mapper and the reducer classes for the job, informing the Hadoop platform to execute the code on the specified input file(s) and controlling the location where the output files are placed [14].

4.1 Example

Consider the problem of counting the number of occurrences of each word in a large collection of documents. The user would write code similar to the following pseudocode.

```
map(String key, String value):
  // key: document name
  // value: document contents
    for each word w in value:
    EmitIntermediate(w, "1");
reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
    int result = 0;
    for each v in values:
      result += ParseInt(v);
  Emit(AsString(result));
```

The map function emits each word plus an associated count of occurrences (just '1' in this simple example). The reduce function sums together all counts emitted for a particular word. In addition, the user writes code to fill in a mapreduce specification object with the names of the input and output files, and optional tuning parameters. The user then invokes the MapReduce function, passing it the specification object. The user's code is linked together with the MapReduce library (implemented in Java) [16].

4.2 Types

Even though the previous pseudocode is written in terms of string inputs and outputs, conceptually the map and reduce functions supplied by the user have associated types.

map (k1,v1) →list(k2,v2)
reduce (k2,list(v2)) →list(k3,v3).

That is, the input keys and values are drawn from a different domain than the output keys and values. Furthermore, the intermediate keys and values are from the same domain as the output keys and values. Finally we can say the Input and Output types of a MapReduce job as:

(input) <k1, v1> -> **map** -> <k2, v2> -> **combine** -> <k2, v2> -> **reduce** -> <k3, v3> (output).

5. Problem Formulation

MapReduce over traditional DBMS: Our traditional DBMSs have adopted such strategies which are not appropriate for solving extremely large scale data processing tasks. There has been a need for some special purpose data processing tools that are adapted for solving such problems[17]. While MapReduce is referred to as a new way of processing big data in data center computing, it is also criticized as a "major step backwards" in parallel data processing in comparison with DBMS [18]. However, many MapReduce proponents in industry argue that MapReduce is not a DBMS and such between them is not a just [19]. Although Hadoop won the 1st position in GraySort benchmark test for 100 TB sorting (1 trillion 100-byte records) in 2009, its winning was achieved with over 3,800 nodes. MapReduce or Hadoop would not be a cheap solution if the cost for constructing and maintaining a cluster of that size was considered [20]. Analysis of 10-months of MR logs from Yahoo's M45 Hadoop cluster and MapReduce usage statistics at Google are also available. The studies exhibit a clear tradeoff between efficiency and fault-tolerance. MapReduce increases the fault tolerance of long-time analysis by frequent checkpoints of completed tasks and data replication. However, the frequent I/Os required for fault tolerance reduce efficiency. Parallel DBMS aims at efficiency rather than fault tolerance. DBMS actively exploits pipelining intermediate results between query operators. However, it causes a potential danger that a large amount of operations need be redone when a failure happens. With this fundamental difference, we describe some advantages and limitations of MapReduce framework [5].

6. Advantages of Mapreduce

MapReduce is simple and efficient for computing aggregate. Thus, it is often compared with "filtering then group-by aggregation" query processing in a DBMS. Here are major advantages of the MapReduce framework for data processing. For the huge data processing task, the key advantages of the MapReduce framework are as [5]:

- **Simple and easy to use**: The MapReduce model is simple but expressive. A programmer defines his task by using only Map and Reduce functions. There is no need for him to specify the physical distribution of his work across nodes.
- **Flexible MapReduce**: It does not have any dependency on data model and schema. A programmer can deal with irregular or unstructured data more easily than they do with DBMS.
- **Independent of the storage**: It is independent from underlying storage layers. Thus, MapReduce can work with different storage layers such as Big Table and others.
- **Fault tolerance**: It is highly fault-tolerant. It is reported that MapReduce can continue to work in spite of an average of 1.2 failures per analysis job at Google.
- **High scalability**: The best advantage of using MapReduce is high scalability. Yahoo! reported that their Hadoop gear could scale out more than 4,000 nodes in 2008.

MapReduce is simple and efficient tool for query processing in a DBMS. Thus, due to these major advantages of MapReduce, we have integrated it with one of the advance technique of data processing from large databases. The increasing interest and popularity of MapReduce has led some relational DBMS vendors to support MapReduce functions inside the DBMS. This capability not only offers the benefits outlined above for deploying user defined functions, but also adds the advantages of MapReduce to the relational DBMS environment, i.e., the ability to process multi-structured data using SQL. It also brings the maturity of relational DBMS technology to MapReduce processing. The Teradata Aster Database is an example of a product that supports MapReduce [18].

7. Discussion And Challenges

MapReduce is becoming ubiquitous, even though its efficiency and performance are controversial. There is nothing new about principles used in MapReduce. However, MapReduce shows that many problems can be solved in the model at scale unprecedented before. Due to frequent checkpoints and runtime scheduling with speculative execution, MapReduce reveals low efficiency. However, such methods would be necessary to achieve high scalability and fault tolerance in massive data processing. Thus, how to increase efficiency guaranteeing the same level of scalability and fault tolerance is a major challenge. The efficiency problem is expected to be overcome in two ways: improving MapReduce itself and leveraging new hardware. How to utilize the features of modern hardware has not been answered in many areas. However, modern computing devices such as chip-level multiprocessors and Solid State Disk can help reduce computations and I/Os in MapReduce significantly. The size of MR clusters is continuously increasing. A 4,000-node cluster is not surprising any more. How to efficiently manage resources in the clusters of that size in multi-user environment and achieving high utilizations of MR clusters is also challenging [5].

8. Conclusion

We are living in the big data era where enormous amounts of heterogeneous, semi structured and unstructured data are continually generated at unprecedented scale, and processing large volumes of data has never been greater. Through better Big Data analysis tools like Map Reduce over Hadoop and HDFS, guarantees faster advances in many scientific disciplines and improving the profitability and success of many enterprises. MapReduce has received a lot of attentions in many fields, including data mining, information retrieval, image retrieval, machine learning, and pattern recognition. However, as the amount of data that need to be processed grows, many data processing methods have become not suitable or limited.

In this paper we discussed about the MapReduce framework for efficient analysis of big data and for solving challenging data processing problems on large scale datasets in different domains. MapReduce provides a simple way to scale your application. It effortlessly scales from a single machine to thousands, providing fault tolerant & high performance.

References

[1] R. Gupta, "Journey from Data Mining to Web Mining to Big Data," *Int. J. Comput. Trends Technol.*, vol. 10, no. 1, pp. 18–20, 2014.

[2] S. Maitrey and C. K. Jha, "An integrated approach for CURE clustering using map-reduce technique," *Proc. Elsevier, ISBN 978-81- 910691-6-3,2nd*, pp. 630–637, 2013.

[3] A. Bifet, "Mining big data in real time," *Inform.*, vol. 37, no. 1, pp. 15–20, 2013.

[4] S. Lenka Venkata, "A Survey on Challenges and Advantages in Big Data," vol. 8491, pp. 115–119, 2015.

[5] S. Maitrey, C. K. Jha, and C. K. Jha, "Handling big data efficiently by using map reduce technique," *Proc. - 2015 IEEE Int. Conf. Comput. Intell. Commun. Technol. CICT 2015*, pp. 703–708, 2015.

[6] A. Bechini, F. Marcelloni, and A. Segatori, "A MapReduce solution for associative classification of big data," *Inf. Sci. (Ny).*, vol. 332, pp. 33–55, 2016.

[7] S. Maitrey and C. K. Jha, "MapReduce: Simplified Data Analysis of Big Data," *Procedia Comput. Sci.*, vol. 57, pp. 563–571, 2015.

[8] J. Dean and S. Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters," *Proc. 6th Symp. Oper. Syst. Des. Implement.*, pp. 137–149, 2004.

[9] K. U. . Jaseena and J. M. David., "Issues, Challenges, and Solutions: Big Data Mining," *Comput. Sci. Inf. Technol.*, pp. 131–140, 2014.

[10] V. Prajapati, *Big Data Analytics with R and Hadoop*, First publ. BIRMINGHAM - MUMBAI: Packt Publishing Ltd., 2013.

[11] A. B. Patel, M. Birla, and U. Nair, "Addressing big data problem using Hadoop and Map Reduce," *3rd Nirma Univ. Int. Conf. Eng. NUiCONE 2012*, pp. 6–8, 2012.

[12] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. Ullah Khan, "The rise of 'big data' on cloud computing: Review and open research issues," *Inf. Syst.*, vol. 47, pp. 98–115, 2015.

[13] D. A. Heger, "Hadoop Design , Architecture & MapReduce Performance," *(DHTechnologies - www.dhtusa.com*, pp. 1–18, 2011.

[14] Dr.Siddaraju, C. L. Sowmya, K. Rashmi, and M. Rahul, "Efficient Analysis of Big Data Using Map Reduce Framework," *Int. J. Recent Dev. Eng. Technol.*, vol. 2, no. 6, pp. 64–68, 2014.

[15] S. Suryawanshi and P. V. S. Wadne, "Big Data Mining using Map Reduce : A Survey Paper," *www.iosrjournals.org*, vol. 16, no. 6, pp. 37–40, 2014.

[16] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, p. 107, 2008.

[17] D. Florescu and D. Kossmann, "Rethinking cost and performance of database systems," *ACM SIGMOD Rec.*, vol. 38, no. 1, p. 43, 2009.

[18] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," *Proc. 35th SIGMOD Int. Conf. Manag. data*, pp. 165–178, 2009.

[19] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, "MapReduce and parallel DBMSs," *Commun. ACM*, vol. 53, no. 1, p. 64, 2010.

[20] S. Loebman, D. Nunley, Y. C. Kwon, B. Howe, M. Balazinska, and J. P. Gardner, "Analyzing massive astrophysical datasets: Can Pig/Hadoop or a relational DBMS help?," *Proc. - IEEE Int. Conf. Clust. Comput. ICCC*, 2009.

[21] http://www.thegeekstuff.com/2012/01/hadoop-hdfs-mapreduce-intro .

[22] http://sci2s.ugr.es/BigData .