TUNISIAN REPUBLIC
MINISTER OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH

Master Thesis

Submitted in order to obtain the Research Master Degree in:

Business Intelligence

"Informatique Décisionnelle et Intelligence Appliquée à la Gestion"

Queries placement problem

Presented by

Dhouha JEMAL

Supervised by

Pr. Rim FAIZ

Pr. Ladjel BELLATRECHE

Academic Year 2013-2014

Queries placement problem

Dhouha Jemal

Abstracts

Data is the most precious asset of companies. Specific to their business sector either appropriate to their own customers or competitors, the data can be mainspring of competitiveness and innovation. This explains the importance granted to the selectivity of data processing tools.

With the data volume which does not stop growing and the multitude of sources which led to of structures diversity, the classic tools of data management became unsuitable for processing. Hence the rapid development and change of the databases world, the evolution of data management solutions and the imposition of the Big date in our technological landscape which reflects both the data explosion and the recent capacity to handle it. This data management systems diversity presents a difficulty in choosing the best solution to interpret, protect and manage data according to the user's needs while preserving data availability.

In this work, we propose two contributions: The first is an implementation and a refinement of a cost model for MapReduce paradigm; Then, we propose an hybrid approach between two main categories of data management systems: classic DBMSs and NoSQL DBMSs. The idea is to integrate the ORDBMS PostgreSQL and MapReduce to perform OLAP queries in a goal of minimizing Input/Output costs in terms of the amount of data to manipulate, reading and writing throughout the execution process.

In order to evaluate our proposed approach performance, we start by assess the Input/Output costs calculated by the refined MapReduce process cost model compared real Input/Output costs. Then, we valid the proposed approach through experiments showing the significant gain of the cost saved up compared to executing OLAP queries independently on MapReduce and PostgreSQL.

Key words : MapReduce, RDBMS, NoSQL, integration, hybrid, cost, performance, OLAP .

Dedications

Believing that nothing impossible as long as Allah is on our side,

I dedicate this thesis to my beloved parents,

my brothers and sisters for believing in me and encouraging me during all of my years of study.

I also dedicate it to all of my dear friends for their support.

Dhouha

Acknowledgments

My sincere gratitude goes to **Pr. Rim FAIZ**, my master thesis supervisor for her encouragement and guidance that truly helped the progression of my research internship. I am also thankful for her availability, supervision, constructive criticism and valuable advices, allowing me achieving and improving this work.

The special thank also goes to **Pr. Ladjel BELLATRECHE**, the co-supervisor of this master thesis at University of Poitiers, for accepting me among the team and inspiring me throughout this research. I also thank him for his uninterrupted encouragement, time, and efforts.

I would like to thank **Ahcene BOKORCA,** for his good advice and friendship. He has been invaluable on both an academic and a personal level, for which I am extremely grateful.

With immense honor, I thank all of the LARODEC Tunis and LIAS Poitiers members for their help during my internship.

Finally, it is with gratitude that I thank all the jury members for agreeing to evaluate my work and all the professors for the knowledge and skills they gave me during all my years of study.

Contents

List of Figures

List of Tables

Acronym

BI	Business Intelligence
DB	DataBase
DBMS	DataBase Management System
ETL	Extract-Transform-Load
Hadoop	High-availability distributed object-oriented platform
HDFS	Hadoop Distributed File System
IS	Information System
IT	Information Technologie
MR	MapReduce
NoSQL	Not Only SQL
OLAP	OnLine Analytical Processing
OLTP	OnLine Transaction Processing
OODBMS	Object-Oriented DataBase Management System
ORDBMS	Object Relational DataBase Management System
RDBMS	Relational DataBase Management System
RID	Row IDentifier
SQL	Structured Query Language

Introduction

The data is currently in the middle of themes, and it is growing at an alarming speed in both volume and structure. The data explosion is not a new phenomenon. It is just accelerated in an incredible way and has an exponential number of technical and application challenges.

It's not easy to measure the total volume of data generation and processing. Data generation is estimated of 2.5 trillion bytes of data every day[1]. This impressive figure masks even more important evolutions. First, unstructured data will grow faster than structured data. Moreover, beyond the storage, challenges will focus on the capacity to process the data and make it available to users. As described in (Ordonez, 2013), with the worldwide volume of data which does not stop growing, the classical tools for data management have become unsuitable for processing. New technologies such as (Furtado, 2009; McClean *et al.*, 2013) are necessary to answer the explosion of the volume of data that it is a question of storing but also of making accessible and of analyzing.

Taking into account new treatment needs proposed in (Cuzzocrea *et al.,* 2013), the database world has been evolved. Then it is continuously required to find up a way to escape the limitations of data process such as lack of performance, resource limits, and fault tolerance, and open issues and challenges are raised on data processing tools. This explains the fast evolution of the data management systems and the

[1]http://www-01.ibm.com/software/fr/data/bigdata/ *September 2014*

orientation towards the parallel architectures (Furtado, 2009; Stonebraker *et al.*, 2010; DeWitt and Gray, 1992) with the aim of achieving the highest number of transactions in a smaller possible time in order to cope with the explosion in the volume of data.

The implications of the rise of data generation challenge the needs of data processing, as presented in (Besse *et al.*, 2014). But the impact of data abundance extends well beyond volume. With the exponential generation, sources of data changes producing new types of data and content which gave rise to a new challenge in the treatment of unstructured data. In this context, a new technological field has emerged: the Big Data (Narasimhan and Bhuvaneshwari, 2014). In this age of explosion in the volume of information and structures diversity, Big Data aims to provide an alternative to traditional solutions database and analysis.

With the new challenges in data processing described in (Cuzzocrea *et al.,* 2013) and the emergence of big data, the production environment for analytical data management applications is changing. This will have an impact on companies which, as proposed in (Furtado, 2009), augment their existing data management capabilities with technology that enables new analytic capabilities to improve the health of the business . The main objective of companies is to have a general view of the activity treated, anticipates the actions, be in tune with the expectations of its customers and can thus adapt the right strategies, and facilitate decision-making and decision analysis leaders for informed management of the company. This explains the increasing strategic importance which companies grant to tools to interpret, protect and manage data according to their strategic value and its sensitivity degree while preserving their availability.

The question is how organizations should prepare for these developments in the

big data ecosystem, how to use information and supporting, what technology to use for data analysis in such an environment, and what are best practices in managing enterprise for best performance and optimize costs.

Therefore, it is important to benefit the diversity of the solutions proposed for the data analysis. This diversity of technologies can be classified into two main categories (Mchome, 2011): The first is the classical tools of data management with its subcategories such as relational and object-oriented system management or even a combination of both, which present several optimization modes such as indexes and materialized views. The second category is the new model of data management called NoSQL (Not Only SQL) DataBase Management System (DBMS) presented in (Strauch *et al.*, 2011), which is gaining significant attention given its capacity for processing unstructured data.

The aim of this work is to integrate the two categories of data management systems. Within the first category of the classic data management systems, we appoint the sub category of the ORDBMS represented by the tool PostgreSQL (Douglas and Douglas, 2003; Matthew and Stones, 2005). The category of the NoSQL DBMS will be represented by the Hadoop MapReduce platform proposed in (Dean and Ghemawat, 2008), the paradigm which met a big success for the applications that process large amounts of data. The main idea of integration leans on the comparison of query execution costs by both paradigms with the aim of the minimizing the Input/Output costs. In this study, we propose different contributions related to the objective of integration. We offer a refinement of a proposed cost model for a MapReduce process. Then we suggest an approach to optimize the OLAP queries Input/Output execution cost.

Document organization

This document is organized as follows:

- Chapter 1 reviews the background of Big Data and DBMSs as the framework for this work.
- Chapter 2 describes the motivations and aims of this work and presents the essential work of the state of the art in order to solve problems related to this issue.
- Chapter 3 presents a proposed refinement of a Input/Output cost model for Hadoop MapReduce process.
- Chapter 4 describes the mechanism of the proposed approach.
- Chapter 5 validates the proposed approach through experiments and presents the results of experimental evaluations.

We finish by concluding the research work proposed, giving guidelines for future work, and opening questions recently emerging in these areas.

Chapter 1

Data management: between classic DBMSs and Big Data solutions

1.1 Introduction

The migration of a computer application is the passage of an information system (IS) or an application to a new platform. Among the migration types, we find the data migration. This corresponds to the transfer of data from one persistent system to another, in order to reap the benefits of technological and functional evolution of the database management systems.

Previously, we mentioned some details of our problem and contributions. Before going into deep details, we introduce in this chapter a general overview of the framework.

This chapter is organized as follows: section 1.2 presents the Big Data field. Section 1.3 defines the Datawarehouses. Sections 1.4 and 1.5 describe two DBMS's main categories NoSQL and classic DBMSs.

1.2 Big Data era: definition, caracteristics and applications

The term Big Data, explained by (Narasimhan and Bhuvaneshwari, 2014), was raised

the first time by the Gartner office[1] in 2008. It refers to the explosion of data volume and new technological capabilities offered to answer it. Big data can be defined as "a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data, by enabling high-velocity capture, discovery, and/or analysis"[2].

A strong interest towards the term "Big Data" is arising in the literature actually. Though scalable data management has been a vision for more than three decades and much research has focussed on large scale data management in traditional enterprise setting, Big Data brings its own set of novel challenges that must be addressed to ensure the success of data management solutions. This field has received much attention from across the computing and research community, and a lot of work has been done in this context such as (Dean and Ghemawat, 2008; Cohen *et al.,* 2009; Agrawal *et al.,* 2011; Wang and Chan, 2013). The goals of the big data solutions are to meet the new challenges of treating very important volume of structured and unstructured data, located on various terminals.

As presented in (Cuzzocrea *et al*., 2013) Big Data repositories have two intrinsic factors: (i) size, which becomes really explosive in such data sets; (ii) complexity which can be very high in such data sets. Every day, the amount of data created and manipulated is increasing. all the sectors of activity are affected by this phenomenon. This exponential growth is due to several factors such as trends in the number of users of IT (Information Technology) solutions and data generation by machines. These masses of data bring larger and finer opportunities for analyzes as well as new uses of the information. Data today comes from multiple sources such as business transactions and social networks, and in all types of formats: Structured numeric data in traditional databases, and unstructured such as text documents, email, video,

[1] http://www.gartner.com/technology/home.jsp *August 2014*

[2] http://idc-cema.com/eng/events/54792-idc-big-data-and-business-analytics-forum-2014 *September 2014*

audio and financial transactions. Managing, merging and governing both explosing amount and different varieties of data is something many organizations still grapple with.

The web and social networks, whether they are open to all or developed in a professional context, provide kind of opportunities for big data. As described in (Sagiroglu and Sinanc, 2013), McKinsey Global Institute in (Manyika *et al.*, 2011) specified the potential of big data in five main topics:

- Healthcare (clinical decision support systems, analyze disease patterns, improve public health).
- Public sector (discover needs, decision making with automated systems to decrease risks, innovating new products and services).
- Retail (in store behavior analysis, variety and price optimization, product placement design, web based markets).
- Manufacturing (developed production operations, supply chain planning).
- Personal location data (smart routing, geo targeted advertising or emergency response).

In this context, Big Data approach aims to provide an alternative to traditional solutions database and analysis. Big data solutions add some features to classics DBMSs in order to satisfy new data management needs in a new ecosystem of explosive volume of structured and unstructured data. Actual research trends in the field of Data Warehousing and OLAP over Big Data are rising, such as (Cuzzocrea *et al.*, 2013). The next section presents the general architecture of a data-warehouse.

1.3 Data-warehouses: foundation of a decision support system

as proposed in (Furtado, 2009), data warehouses (DW) are at the core of such systems. In this context, we explain in this section the general architecture of a data-warehouse to differentiate it from traditional databases.

The data-warehouse is a database that collects information from various enterprise information systems, for analysis and reporting activities. According to the definition of Bill Inmon (Inmon, 2005) we show the characteristics of a data-warehouse as described by(Gangarski and Doucet, 2001; Franco, 1997) as follows:

- Subject-oriented: the data collected must be oriented "business ". So, they are organized by theme rather than by application.
- Integrated: to ensure consistency of information, data from various sources must be integrated before storage in the data warehouse.
- Nonvolatile: data, unlike those of traditional databases, should be ongoing. Thus, a refresh of the data warehouse should be able to add new data without modifying or lose existing ones.
- Historized: it is essential that the data will be dated to consider their evolution in decision-making and policy analysis since they are based on the entire history of past trends to predict future ones planned.

The data-warehouse is the foundation of a decision support system for an organization. Unlike a taditional database supporting transactional queries like OLTP (On-Line Transaction Processing), a data warehouse is designed to support multidimensional OLAP queries (On-Line Analytical Processing). The table 1.1 illustrates well this difference.

Table 1.1: Databases Vs Datawarehouses

	Operating systems	Data Warehouses
Design	Application oriented	Subject oriented
Design principle	Relational design	Multidimensional design
Goal	Management & Production	Planning & Analysis
Data type	Recent, Detailed	Historised, Aggregated
user interaction	Interrogation, Update	Interrogation
Queries	Simple, Predetermined	Complexe, Ad-hoc
Transaction	Short, Real-time	Long
Size	Several Gigabytes	Several Terabytes

As mentioned in (Cuzzocrea *et al.,* 2013), several research problems arise when computing OLAP data cubes over Big Data. On the other hand, as proposed by (Ramakrishnan *et al.,* 2013), it is often difficult to attach schema, or meaning, to the data beforehand and the schema and meta-data tends to evolve over time, data-stores need to accommodate real-time changes to the structure of the data while making the data searchable on any attribute. Thus, applications are increasingly using schema-less data-stores also referred to NoSQL databases which are presented in the next section.

1.4 NoSQL DBMSs

NoSQL (Not only SQL) Database management system, as presented in (Strauch *et al.*, 2011), appoints a category of DBMSs based on non-relational distributing architecture.

One of the main strengths points of the NoSQL database is its performance (Nance *et al.,* 2013). It explains that many Web giants like Facebook[3], Twitter[4] and LinkedIn[5] chose to migrate their data on it. The advantages of NoSQL are at least three: Coherence (visibility by all nodes in a system of the identical data

[3]www.facebook.com *October 2014*
[4]www.twitter.com *October 2014*
[5]www.Linkedin.com *October 2014*

at the moment T); data availability even in case of failure; ability to partition distributed system. NoSQL databases are increasingly used in big data and real-time web applications.

There are currently many NoSQL solutions such as Hadoop[6], HBase (Carstoiu, 2010; George, 2011) and Neo4j (Eifrem, 2009). This diversity presents a difficulty in choosing the best solution, especially for the deployment of a database and data processing. There are four main families of NoSQL DBMS, each brings a different data representation, has specificities and simplifies the manipulation of some data type.

- Key-value oriented: The simplest representation, means a single correspondence between a key and a value such as Redis[7].
- Document oriented: Based on the key value-oriented family, except that the value is represented in the form of a document such as XML or JSON such as MongoDB[8] (Sabharwal *et al.*, 2014).
- Columns oriented: Another evolution of key-value model, it allows to have a large number of values in a line, so allowing storing the relation one-to-many type. Unlike the key-value model which allows querying by value such as HBase (Carstoiu, 2010).
- Graph oriented: Allows modeling, storage and manipulation of complex data connected by non-trivial or variables relations such as Neo4j (Eifrem, 2009).

The figure 1.1 illustrates these different representations.

[6]http://hadoop.apache.org/ *April 2014*
[7]http://redis.io/ *April 2014*
[8]http://www.mongodb.org/ *April 2014*

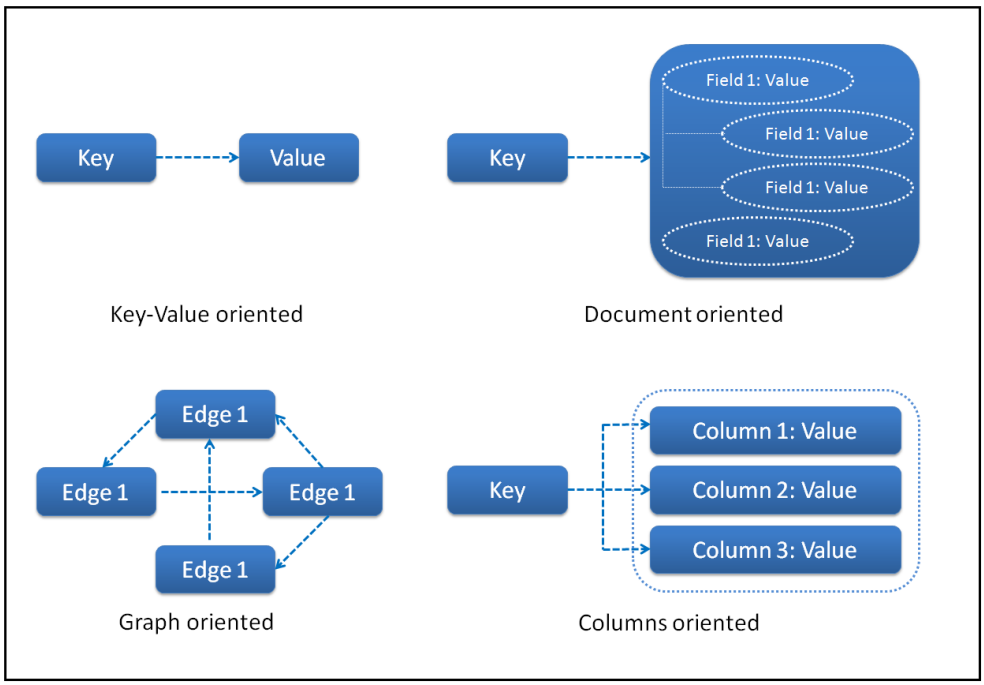

Figure 1.1: Types of NoSQL DBMSs

In this context, MapReduce (MR) paradigm met a big success for applications that process large amounts of data. It was initially proposed by Google to facilitate the development of web search applications on a large number of machines. In the next section, we present this framework as described by (Dean and Ghemawat, 2008).

The era of Big Data recently has arrived due to the explosion in the volume of data. To keep up with the times, Hadoop (High-availability distributed object-oriented platform) and its various related projects presented in (Narasimhan and Bhuvaneshwari, 2014) have emerged as a solution for efficient analysis and processing of Big Data (Shvachko *et al.*, 2010). Handling with distributed data in a cluster requires parallel computing techniques; the most known technique is MapReduce the new paradigm presented in (Deann and Ghemawat, 2008). This involves dividing the data to be processed in independent partitions, treat these partitions in parallel

and finally combining the results of these treatments.

Hadoop[9] is the Apache Software Foundation open source and Java-based implementation of the MapReduce framework. MapReduce is a programming model introduced by Google for processing very large data-sets. One who works in the world of databases and NoSQL, he certainly heard of MapReduce the powerful tool characterized by its performance for heavy processing to be performed on a large volume of data that it can be a solution to have the best performance hence makes it very popular with companies that have large data processing centers such as Amazon and Facebook[10].

Although the MapReduce framework has found great success in analyzing and processing large amounts of data on large clusters, its implementation in Google unfortunately is not free. While Hadoop MapReduce as mentioned in (White, 2009), is a Java open source implementation of MapReduce in HDFS (Hadoop Distributed File System) that will be presented in the next section.

1.4.1 HDFS: Hadoop Distributed File System

Hadoop is designed to scale up from single servers to thousands of machines, each offering local computation and storage. As described in (White, 2009), it proposes a distributed storage system, Written in Java, via its file system HDFS (Hadoop Distributed File System). MapReduce officiates the file system HDFS to perform processing on large data volumes.

The distributed storage infrastructure store very large volumes of data on a large number of machines, and manipulate a distributed file system as if it were a single hard drive. The Hadoop MapReduce framework may be run in a cluster of large number of machines which follows a master/slave architecture. When the data is

[9]http ://hadoop.apache.org *April 2014*
[10]www.facebook.com

loaded on the cluster, it is distributed to all the nodes of the cluster. A single node, called NameNode constitutes the master server that manages the file system namespace, It maintains the file-system tree and meta-data for all directories and files and manage the access to files. The rest of the nodes are the slaves, called DataNodes stores data locally on the machine they run on with a periodic report back to the NameNode with the storage state.

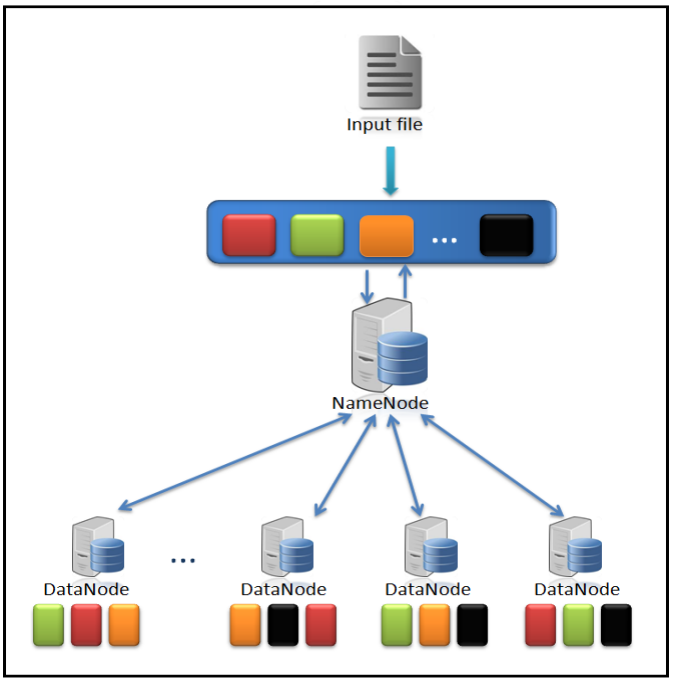

Figure 1.2: Hadoop Distributed File System

As illustrated in the figure 1.2, HDFS deals with data in blocks. When an input file is loaded on the cluster, it is split into one or more blocks according to a predefined size (64 MB by default) and the NameNode will be responsible for allocating blocks within Datanodes. In order to prevent data loss, each block will be replicated across several machines to overcome a possible problem of a single machine failure. In addition, HDFS follows data locality optimization strategy. This strategy aims to prevent unnecessary network transfers through the NameNode which tries to assign

the processing of each block to the DataNode on which the data block is actually stored.

1.4.2 MapReduce process

In this section we describe the MapReduce process as presented in (Dean and Ghemawat, 2008; Lee *et al.*, 2011; Lammel, 2008). MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster. It was created by Google in 2003, in order to simplify parallel processing and distributed data on a large number of machines with an abstraction that hides the details of the hardware layer to programmers. The MapReduce model consists of two primitive functions: Map and Reduce. Two main components manage the Map/Reduce process:

- Job tracker: assigns tasks to the Tasktrackers to be performed.
- Task tracker: accepts tasks (map and reduce) from the job tracker.

The Reduce phase: The reduce tasks are then run over the resulting data, to combine the outcome of the Map Results. During each reduce phase, the reducer the workers that implement the reducers, retrieve the intermediate data from the mappers. In details, each reducer fetches the corresponding partition from For reasons of data locality optimizing the NameNode and JobTracker are co-located in the same node that is the Master and the same for the DataNode and task tracker constituting a Slave also known as Worker. Each worker on the cluster may be assigned the role of a mapper or a reducer according to the job that executes. The map and reduce functions are defined by the user, and will be implemented by the mapper and the reducer correspondingly.

The Map phase: During each map phase, the mapper reads the input block and converts each record into a Key/Value pair. The user defined map function transforms each pair into a new Key/Value pair based on the user's implementation. The subsequent phase consists of partitioning/grouping and sorting the map function's outputs.

The Reduce phase: The reduce tasks are then run over the resulting data, to combine the outcome of the Map Results. During each reduce phase, the reducer the workers that implement the reducers, retrieve the intermediate data from the mappers. In details, each reducer fetches the corresponding partition from each mapper. Each set of partitions will be merged constructing pairs of Key/List(Values) based on the same key. The new intermediate pairs of Key/List(Values) are combined based on the user's define function to return a new key/value pair. The output pairs are stored on the HDFS in the output file.

We will explain the process of a MapReduce job and the concepts of map and reduce through the example of words counter frequently used:

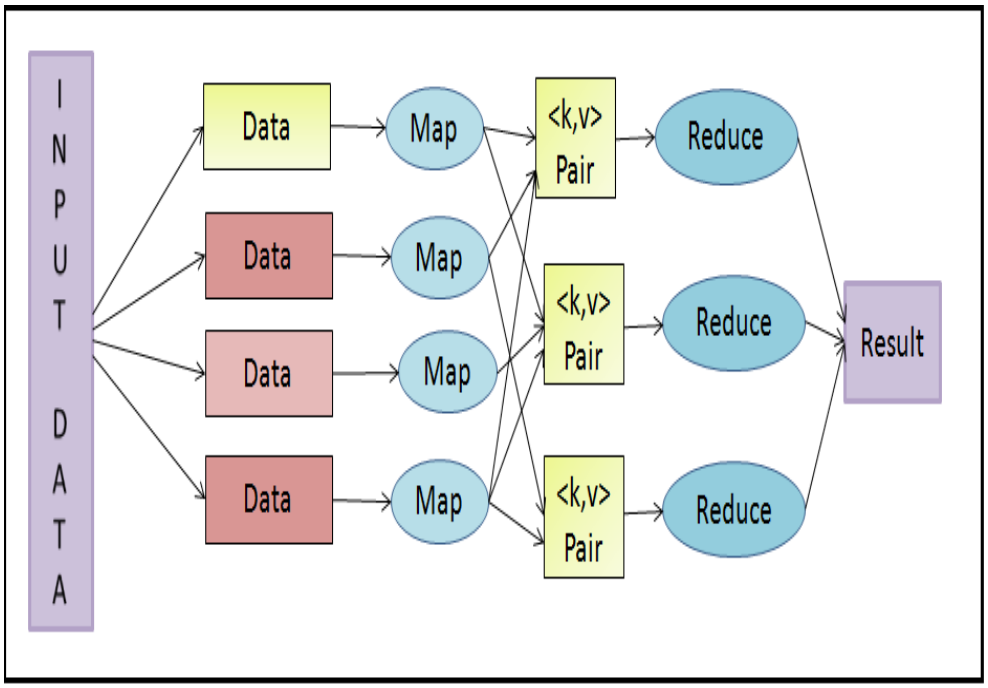

Figure 1.3: MapReduce job process

In the example, we are going to unwind one job MapReduce to count the number of words contained in the first line shown in bold face. The user wishes to count all the words with the exception of the word 'is'.

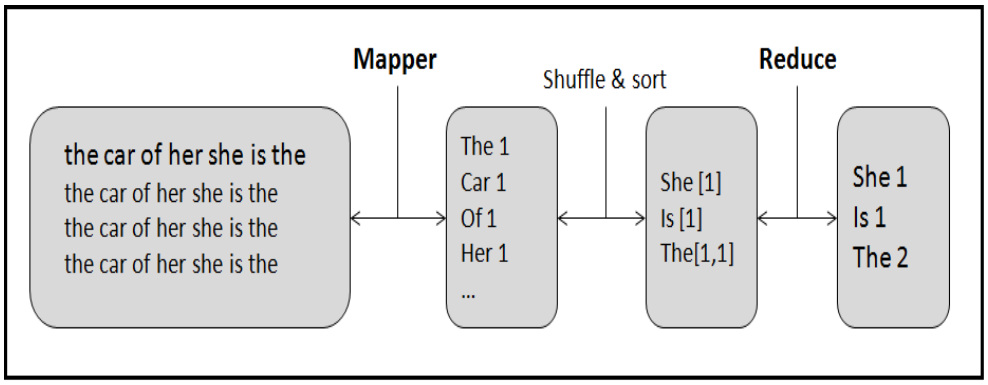

Figure 1.4: MapReduce job example

Map: The map function is written as follows: map (key1, value1) → List (key2,

value2). Using a key-value pair, the map function returns a set of new key-value pairs. First it cut the line in several words and returns a list of key-value pairs, where each key is the word, and each value is "1". During this phase, the user can filter words considered little interesting, as 'is'. In our example, the input key is the line number in the file and the value is "the car of her she is the". The result of the map function is given below.

```
(the, 1)
(car, 1)
(of, 1)
(her, 1)
(she, 1)
(is, 1)
(the, 1)
```

Before presenting the reduce function, two intermediate transactions must be performed to prepare the value of its input argument. The first operation called shuffle allows grouping the values the key of which is common. The second operation called sort allows sorting by key. Thus, after performing the duties shuffle and sort the result of the example is as follows.

```
(the, [1, 1])
(car, [1])
(of, [1])
(her, [1])
(she, [1])
```

Reduce: the reduce function is written as follows: reduce (key2, List (value2)) $\rightarrow$ List (value2). It performs the user defined function, which in this example is to add the values of the same key. The result for reduce is the following.

```
(the, 2)
(car, 1)
(of, 1)
(her, 1)
(she, 1)
```

1.5 Classic DBMSs

DBMS stands for Database Management System. It is a software system that used to store and to manage data. It also defines rules to validate and manipulate this data. By managing user requests, it receives instruction and accordingly instructs the system to make the necessary changes to let users create and access data in a database. Such systems support SQL (Structured Query Language) language and use the relational data model, with some variety to support distributed applications. DBMSs present many different types.

1.5.1 RDBMSs

RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL, and is a type of database management system (DBMS) that is based on the relational model that stores data in the form of related tables. So that, an important feature of relational systems is that a single database can be spread across several tables. RDBMSs have become a predominant powerful choice for the storage of information in new databases because they are easier to understand and use. It is based on relation between data what makes it powerful because the possibility of viewing the database in many different ways since the RDBMS require few assumptions about how data is related or how it will be extracted from the database.

1.5.2 ODBMSs

OODBMS stands for Object-Oriented Database Management System (sometimes shortened to ODBMS for Object Database Management System). As presented in (Ketabchi *et al.*, 1990), it has been considered since the early 1980s, is the result from integration of database capabilities with object programming language capabilities, so that is a database management system (DBMS) that supports the modeling and creation of data as objects and information is represented in the form of objects as used in object-oriented programming. Some object-oriented databases are designed to work well with object-oriented programming languages, others have their own programming languages.

1.5.3 ORDBMSs

ORDBMS (Object Relational Database Management System) is the result from merge of RDBMS and ODBMS. As described by (Brown, 2000), it provides a middle ground between relational databases and object-oriented databases, extends the relational model and puts an object oriented front end on a relational database (RDBMS). So that this system supports objects, classes and inheritance in database schema and query language. In addition, in an ORDBMS data is manipulated using queries in a query language. Object-Relational Database Management Systems grew out of research that occurred in the early 1990s.

The basic goal presented in (Sabàu, 2007) for the Object-relational database is to bridge the gap between relational databases and the object-oriented modeling techniques used in programming languages. The most notable research project in this field is Postgres (Berkeley University, Californie); Illustra and PostgreSQL are the two products tracing this research[11].

[11] http://en.wikipedia.org/wiki/Object-relational_database *July 2014*

1.6 Conclusion

In this chapter, we presented briefly the background related to our research framework. This included Big Data, Data warehouses and DBMSs. We mentioned how important became this area and how the research topic is very highlighted. This domain faces several challenges which we will discuss in the next chapter.

Chapter 2

Problem description and state of the art

2.1 Introduction

Day after day, zillions of data is generated all over the universe. Many factors contribute to the increase in data volume. The implications of the rise of data generation challenge the needs of data processing, which explains the technological development and diversity of the proposed data management solutions and and explosive growth, both in the number of products and services offered and in the adoption of data analysis technologies in the past two decades.

Information is the heart of business. Companies are constantly based on the data in their systems for decision-making. More accurate analyzes may lead to more confident decision making. And better decisions can mean greater operational efficiencies, cost reductions and reduced risk.In a highly competitive world where innovation makes the difference, a better management of its data and solutions is the best guarantee for businesses. This explains the priority role for technology selectivity to benefit of the solutions proposed for the data analysis and to satisfy the needs of data processing.

In this context, we are going to detail in this chapter challenges of the Big Data field, then the main problems about diversity of data management solutions. And we

finish by giving some previous works who attempted to deal with these bottlenecks.

2.2 Big Data challenges

Nowadays, Big data and its analysis are at the center of modern science and business. It requires a revolutionary step forward from traditional data analysis. As mentioned in (Sagiroglu and Sinanc, 2013), the term Big Data is for massive data sets having large, more varied and complex structure with the difficulties of storing, analyzing and visualizing for further processes or results.

Big Data, as presented in (Sagiroglu and Sinanc, 2013), is characterized by three main components, the 3 V's: Volume, Velocity and Variety.

1. **Volume.** It is the first feature brought by the term "big". The size, which can become a real bottleneck from practical applications, refers to the vast amounts of data generated every second. The volume of data stored today is booming. This amount of data that is being collected daily presents immediate challenges for businesses. we can Just think of social media messages going viral in seconds, the speed at which credit card transactions are checked for fraudulent activities, or the milliseconds it takes trading systems to analyze social media networks to pick up signals that trigger decisions to buy or sell shares.

2. **Velocity.** Speed of data in and out; describes the frequency at which data are generated, captured and shared. Growing flows of data must be analyzed in real time to meet the needs of chrono-sensitive processes. Reacting fast enough and analyzing the streaming data is troubling to businesses, with speeds and peak periods often inconsistent. Bid Data approach opens the possibility to integrate data streams and generate results or data visualization in (almost) real time.

3. **Variety.** The volume of Big Data puts data centers in front a challenge: the variety of data. It's not traditional relational data, this data is raw, semi-structured or unstructured. In fact, 80% of the world's data is now unstructured[1], and therefore can't easily be put into tables (think of photos, video sequences or social media updates). Big Data is in the form of structured and unstructured data. The structured data types are ready for insertion into a database, while unstructured types have an implicit and irregular structure, and not a fixed pattern (non-relational). With big data technology we can now harness differed types of data (structured and unstructured) including messages, social media conversations, photos, video or voice recordings and bring them together with more traditional, structured data.

In (Narasimhan and Bhuvaneshwari, 2014), we consider two additional dimensions when thinking about big data:

1. **Veracity.** Accuracy of collected data is a key feature. As mentioned in (Cuzzocrea *et al.,* 2011) very often, data sources, storing data of interest for the target analytic processes, such as web and social networks are strongly heterogeneous and incongruent. Big Data becomes bigger and the multiple sources of big data are ever increasing. So, build confidence in the Big Data represents a significant challenge due to the possibility of inconsistency and abnormality in the Data. Very large data volumes and multiple heterogeneous sources amplify the need for rigor in the collection and crossing data to remove data uncertainty to build confidence and ensure the security and integrity of data.

2. **Value.** Big Data is gradually transforming organizations around the valuation

[1]https://www.linkedin.com/today/post/article/20140306073407-64875646-big-data-the-5-vs-everyone-must-know *October 2014*

of information. Big Data approach is designed to achieve the strategic objectives of value creation for the company.With the Big Data approach, the data which have not big interest when it are taken isolation, can take a meaning when considered globally. A big data strategy gives businesses the capability to better analyze data with a goal of accelerating profitable growth. Having access to big data, companies generate value from data.

Big data and its analysis are at the center of modern science and business. Inspired by this main motivation, (Cuzzocrea *et al.*, 2011) present a number of open problems and actual research trends related to big data analytics, such as: The data Source Heterogeneity and Incongruence, Filtering-Out Uncorrelated Data, Strongly Unstructured Nature of Data Sources, High Scalability. In this context, a main challenge that has interested the research community and has been the subject of several works such as (Abouzeid *et al.*, 2009; Gruska and Martin, 2010) is: Combining the Benefits of RDBMS and NoSQL Database Systems. It is one of the more relevant features to be achieved by big data analytic systems. As discussed in (Cattell, 2011), it is necessary to combine the benefits of traditional RDBMS database systems and those of the new generation of NoSQL database systems in order to obtain the critical flexibility feature which refers to the property of covering a large collection of analytic scenarios over the same big data partition.

The question is not any more " can Big Data become a relevant competitive advantage? ", but "How can we exploit the opportunities offered by these solutions to optimize our analysis and decision making process? ".

2.3 Problem description

In a global market where the systems, people and processes are interconnected, data

is a valuable and strategic asset. This explains the importance attributed to the storage and analysis of these data and the selectivity of management techniques. Taking into account new treatment needs, the databases world has evolved and proposed new technologies and solutions for data management. These new systems fall into two main categories: the first is the classics database management systems; The second consists in the solutions proposed by the big data approach.

As presented in (Pavlo *et al.*, 2009), for a long time the DBMSs have been the standard technology for data warehousing, especially with the advent of parallel DBMSs that are a robust, high-performance platform based on the relational model and provides SQL as standard query language.

Faced with the massive growth in the volume of data, despite their evolution, relational databases, which have been proven for over 40 years, have reached their limits. These limits are mainly caused by the constraints ACID (atomicity, consistency, isolation and durability) associated with the relational model, these constraints which represented the strength of this model, cripple response times when the database is distributed over a cluster of several machines to handle larger volumes of data. Relational database systems and decision support tools were not originally designed to handle such an amount and richness of data, and it can quickly become complicated and unproductive for companies to access these masses data with classics tools.Therefore, this new problem has led database management systems called "NoSQL" (Not Only SQL), which have added some features to classics DBMSs in favor of simplicity, performance.

According to (Ordonez *et al.*, 2010), distributed file systems and MapReduce present a new technology for such applications. They showed their performance to manage large amounts of data where DBMSs present limits. This new platform massively parallel of data processing is considered in (Stonebraker *et al.*, 2010) as

the best candidate for complex applications data analysis, because these applications require complex data stream. MapReduce does not require preliminary data schema and works well on unstructured data and rarely-modified, while a DBMS is suitable for transactional queries and data sets that are structured, standardized and continuously modified.

In the context of data migration, which corresponds to the data transfer from one system to another in order to benefit of the technological and functional evolution of the database management systems (DBMS), the designer responsible for establishing migration is often faced with the problem of choice the target technology. In this environment of data explosion and diversity the question is what technology to use for data analysis, how to benefit the data management systems diversity? The most important question in evaluating the needs of an application and whether to use NoSQL databases or relational databases depends on the type of application being written, the type of queries that are expected, and the regularity vs. variability of the data's structure.

As discussed in (Nance *et al.,* 2013), large public and content-centric applications will tend to be best served by NoSQL databases. In contrast, internal line of business applications that are supporting business operations will very often be best served by relational databases and may even be served exclusively by relational databases. It is important to pick the right database technology for the task at hand. The relational database model and the NoSQL database model are each good for specific applications. Depending on what problem the organization is trying to solve, it will determine if a NoSQL database model should be used or if a relational database model should be used.

Several comparative studies have been conducted between MapReduce and RDBMSs such as (Mchome *et al.*, 2011; Ordonez *et al.*, 2010; Pavlo *et al.*, 2009; Stonebraker

et al., 2010). MapReduce has been presented as a replacement for the Parallel Database Management Systems. However, as proposed in (Stonebraker *et al.*, 2010), MapReduce can be seen as a complement to a RDBMS for analytical applications, because different problems require complex analysis capabilities provided by both technologies. Many applications will fall in the middle of that spectrum that may choose to use both relational databases and NoSQL databases or pick one or the other depending on the application.

In this context, the question is how organizations should prepare for these developments in the big data ecosystem, how to benefit from the diversity of available proposed technologies, what technology to use for data analysis in such an environment?

2.4 Related works

A strong interest towards the term "Big Data" is arising in the literature actually. Many research works focuse on this actual research trends in the field. In this context, (Sagiroglu and Sinanc, 2013) presents an overview of big data's content, scope, samples, methods, advantages and challenges, details big data's main components and discusses privacy concern on it. Then, (Narasimhan and Bhuvaneshwari, 2014) provides a brief of the buzz-field called Big Data and cover the components of big data from a Hadoop perspective. This study aims to highlight the field's caracteristics with two additional dimensions, and to provide a thorough understanding of big data and its various components in the Hadoop framework.

In (Cuzzocrea *et al.,* 2011), open problems and actual research trends are highlighted with the aim of providing an overview of state-of-the-art research issues and achievements in the field of analytics over big data, and extend the discussion to analytics over big multidimensional data. This work presents several novel research

directions arising in this field, which plays a leading role in next-generation Data Warehousing and OLAP research. In (Cuzzocrea *et al.*, 2013), open problems in the field of Data Warehousing and OLAP over Big Data are highlighted. This work aims to present challenges to adopt Data Warehousing and OLAP methodologies with the goal of collecting, extracting, transforming, loading, warehousing and OLAPing such kinds of data sets, by adding significant add-ons supporting analytic over Big Data.

Big data and its analysis are at the center of modern science and business. Since Google researchers proposed the Map/Reduce programming model for data-analysis and compute-intensive environments in (Dean and Ghemawat, 2008), a lot of research is focused on this new paradigm in order of evaluation and integration. In this context, in (Palla, 2009) an insight into the MapReduce framework in terms of Input/Output cost is provided. This work aimed to exploit the open source Hadoop implementation of the Map/Reduce framework in order to develop a theoretical cost model evaluate the Input/Output cost induced on each node during the execution of a task, and a thorough analysis of three join implementations under the Map/Reduce data-flow on the other hand.

The survey presented in (Lee *et al.*, 2011) intends to assist the database and open source communities in understanding various technical aspects of the MapReduce framework. In this survey, the MapReduce framework is characterized and its inherent pros and cons are discussed. It introduce its optimization strategies reported in the recent literature and discuss the open issues and challenges raised on parallel data analysis with MapReduce.

On the other hand, many research works aim to apply the ideas from multi-query optimization to optimize the processing of multiple jobs on the MapReduce paradigm by avoiding redundant computation in the MapReduce framework. In this direction, MRShare (Nykiel *et al.*, 2010) has proposed two sharing techniques for a batch of jobs.

The key idea behind this work is a grouping technique to merge multiple jobs that can benefit from the sharing opportunities into a single job. However, MRShare incurs a higher sorting cost compared to the naive technique. In (Wang and Chan, 2013) two new job sharing techniques are proposed: The generalized grouping technique (GGT) that relaxes MRShare's requirement for sharing map output. The second technique is a materialization technique (MT) that partially materializes the map output of jobs in the map and reduce phase.

The Pig project at Yahoo (Olston *et al.*, 2008), the SCOPE project at Microsoft (Chaiken *et al.*, 2008), and the open source Hive project[2] introduce SQL-style declarative languages over the standard MapReduce model, aim to integrate declarative query constructs from the database community into MapReduce to allow greater data independence.

The survey presented in (Ordonez, 2013) explains important research that has enabled analytics on large databases inside a DBMS in order to negate that DBMS is not a good technology to analyze big data, going beyond SQL queries, acting just as a reliable and fast data repository. It argue DBMSs cannot compete with parallel systems like MapReduce to analyze web-scale text data. Therefore, each technology will keep influencing each other. In addition, it proposes long-term research issues, considering the big data analytics trend.

On the other hand, a lot of work has been done to compare the MapReduce model with parallel relational databases and there has been some recent work on bringing together ideas from MapReduce and database systems. In (McClean *et al.*, 2013), broader themes of the paradigms are considered rather than the specific implementations of MapReduce and Parallel DBMS. It will discuss MapReduce and Parallel Database Management Systems as competing and complimentary paradigms

[2]http://hadoop.apache.org/hive/. *April 2014*

with the aim of providing a high-level comparison between MapReduce and Parallel DBMS, in order to provide a selection of criteria which can be used to choose between MapReduce and Parallel DBMS for a particular enterprise application.

In (Nance *et al.,* 2013), the pros/cons of NoSQL are discussed, and NoSQL data modeling techniques are presented. This work propose that the SQL and NoSQL models both have their own set of pros and cons that each business has to identify, and then decide which one is better for their company; or if they should use a combination of both SQL and NoSQL.

In (Stonebraker *et al.*, 2010), the differences in the architectural decisions of MapReduce systems and database systems are discussed in order to provide insight into how the systems should complement one another. This work argues that MapReduce is more like an extract-transform-load (ETL) system than a DBMS, as it quickly loads and processes large amounts of data in an ad-hoc manner. As such, it complements DBMS technology rather than competes with it, since databases are not designed to be good at ETL tasks. Then, it describes what is the ideal use of MR technology and highlights the different MapReduce and parallel DMBS markets.

Several research works have been conducted between MapReduce and RDBMSs in the goal of integration. In (Gruska and Martin, 2010), the two systems RDBMSs and MapReduce considered as complimentary and not competitors.In this work, a taxonomy is provided to characterize several existing integration methods. It propose a classification and characterization of current MapReduce and RDBMS integration technologies and argues the need for interoperability between a RDBMS and MapReduce system.

In (Yui and Kojima, 2013) a database-Hadoop hybrid approach to scalable machine learning is proposed. In this approach, batch-learning is performed on the

Hadoop platform, while incremental-learning is performed on PostgreSQL. The training speed is considered as the main metric for evaluating this work.

The work presented in (Abouzeid *et al.*, 2009) attempted to bridge the gap between the two technologies, that is, parallel databases and Map/Reduce model, suggesting a hybrid system that combines the best features from both. But this work evaluated the model having as a metric the computation time and efficiency.

In (Pavlo *et al.*, 2009), experiments are conducted to evaluate both parallel DBMS and the MapReduce model in terms of performance and development complexity. This work showed that the MapReduce model outperformed in scalability and fault tolerance, but at the same time underlined the performance limitations of the model, in terms of computation time. This lack was explained by the fact that the model was not originally designed to perform structured data analysis.

Big Data has become a very important area and the research topic is very highlighted in this field. Its analysis and its proposed solutions are at the center of business and actual research works. The main subjects and available works present a survey on the Big Data in order to provide an overview of big data's methods, advantages and challenges and detailing big data's main components. The second part of research works aims to optimize the treatment of data processing on the MapReduce paradigm by applying the ideas of multi-query optimization to avoid redundant calculation or introducing SQL-style declarative languages over the standard MapReduce model. In addition, the third part is presented by research works which have been conducted between MapReduce and RDBMSs in the goal of integration.

Our proposed work is at the level of the current MapReduce and RDBMS integration technologies, and it aims to minimize the Input/Output cost for data processing. It evaluates the model having as a metric the implicated Input/Output cost.

2.5 Conclusion

In this chapter, we described the different problems faced in managing data and database systems. We also gave the different approaches and studs proposed in the literature in order to solve these problems.

In the next chapters, we detail our work which includes a refinement of a proposed cost model for MapReduce process and an approach to integrate MapReduce paradigm with an RDBMS.

Chapter 3

MapReduce cost analysis: proposed model and suggested refinement

3.1 Introduction

Organizations look to choose the technology to be used to analyze the huge quantity of data. The question is about the selection criteria to be considered in processing the data while meeting the objectives of the organization. The decision support methods not only provide information but also to choose among several options, depending on criteria. The decision support tools help the decision maker to make a choice in a more transparent and more robust. For this purpose several analytical cost models were developed. A comparison of the cost of data analysis on different technologies can support the decision making process for processing data while minimizing costs.

In this context, we present in this chapter the concept of cost models, then an overview on the MapReduce process to propose finally a refinement of a proposed MapReduce cost model.

3.2 Cost models: definition and parameters

Cost models play a key role in the performance evaluation. they are necessary to measure the costs and benefits of optimization structures in order to choose the

optimum configuration of structures and systems.

3.2.1 Cost model: definition

A cost model can be defined as a mathematical function for estimating the cost of performance according to the assigned resources (Gardarin, 2003). In the field of databases, the cost models are mathematical algorithms to estimate the query execution time. These solutions were used in the first time in the context of optimizing execution. The DBMS vendors have implemented approaches based on cost models (cost-based approach) to design their query optimizers. These approaches take into setting an execution plan for a query and return an estimate of its running time, allowing comparing different execution plans for a given query and selecting the best. A second use of cost models is at the physical design phase in order to select the best structures optimization (materialized views, indexes, partitioning).

Cost models help to figure out the cost for certain activities and processes, which aims to: Choose the best configuration, Tend to maximize performance, Solutions selectivity. It is today the most widely applied methodology for measuring execution cost. The models results are typically necessary to obtain approval to proceed. But the major disadvantage of a cost model lies in the simplifying assumptions that may differ the cost obtained from reality (an overestimated or underestimated cost).

Models typically function through the input of parameters that describe the attributes of the product or project in question and possibly physical resource requirements, and provide as output various resources requirements in cost and time.

3.2.2 Cost model: parameters

The parameters considered by the cost models are generally associated with the logical schema of the database and query loads. In what follows we detail these two

parameters.

- Logical schema parameters: We consider a logic schema LS consists of n tables T = {T_1,T_2, . . . ,T_n}. Two parameters are important for each data table : The number of n-tuple denoted || T_i || and the actual table size measured in bytes, denoted |T_i|. The table 3.1 summarizes the parameters of the logical schema. The size of the logical schema is denoted ||LS||, it is equal to the sum of the sizes of the set of tables T.

$$||LS|| = \sum_{i=0}^{n} ||T_i|| \tag{3.2.1}$$

Table 3.1: The parameters of the logical schema

Parameter	Description
T_i	Dimension table T
n	Number of tables of the logical schema
||T||	The size of a table T
|T|	The size of a table T in bytes (the number of pages storing T)
||LS||	The size of the logical schema

- Queries parameters: The cost of query execution depends on two different costs: the cost of Input/Output and the cost of processing by the processor.As explained in (Boukhalfa, 2009), Most existing models of cost do not include the CPU cost since it is well below the Input/Output cost. The only parameters related to requests to consider are the operations performed by these queries such as selecting, join or aggregation.

The performance of the cost model is based on its dependence on various factors

of the platform used. The cost model, being based on the simulation of the operation of the system we will study carefully MapReduce process.

3.3 MapReduce process: an overview

As presented in (Zaharia *et al.*, 2008) MapReduce is emerging as an important programming model for large-scale data-parallel applications such as web indexing, data mining, and scientific simulation. In (Lammel, 2008), it is proposed that MapReduce serves for processing large data sets in a massively parallel manner. The figure 3.3.1 shows an overview of a process of execution MapReduce.

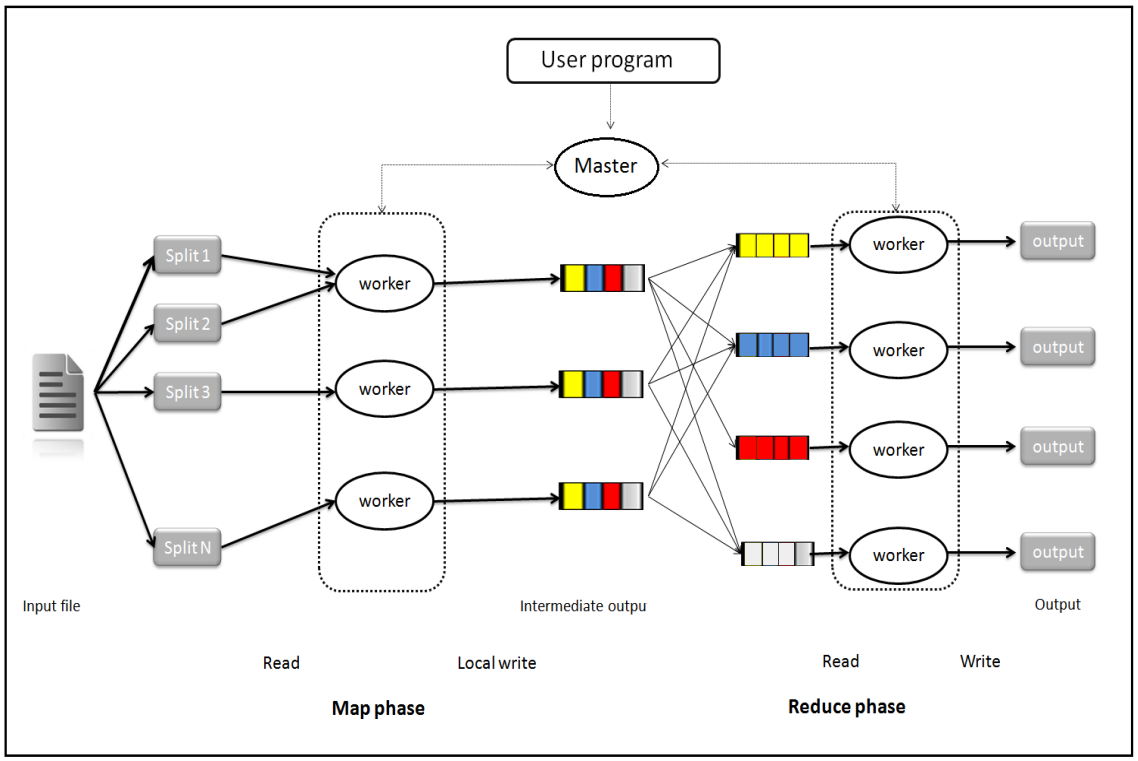

Figure 3.1: MapReduce process overview

The Map/reduce process consists of a sequence of actions/steps that are described as follows:

- When the input data is loaded on the cluster, it is split into N blocks according to a predefined size. The set of blocks is distributed across the nodes. In addition, the framework starts a block replication across several machines to prevent data loss.
- There will be M map tasks that equals to the blocks number. The single node that constitutes the master schedules the map task distribution among the workers following the data locality optimization approach.
- Each worker assigned with a map task reads the records from the corresponding input split. This corresponds to the reading phase of a map task. The mapper converts each record into a Key/Value pair, does the desired transformation based on the user's defined map function and outputs a set of intermediate key-value pairs which will be stored in memory corresponding to the buffering phase.
- When the buffer exceeds a threshold, the intermediate outputs buffered in memory, are periodically spilled to local disk in sorted partitions.
- There will be as many as reduce tasks than partition. Each worker assigned with a reduce task fetches the right partition of data from each map output and writes it in memory.
- The fetched partitions are merged constructing pairs of Key/List(Values) based on the same key.
- The new intermediate pairs of Key/List(Values) are combined based on the user's define reduce function to result a new key/value pair. The output of the reducers is written and stored in the HDFS.

In order to analyze the Input/Output cost of MapReduce process, a study of the architecture of the framework and consider every intermediate step carefully is required. the MapReduce Input/Output cost model is presented in the next section.

3.4 MapReduce process: Input/Ouput cost model

MapReduce is a programming model for performing parallel computations on large

data sets. In the fashion of integrating this paradigm with a RDBMS, we opt to calculate the cost of performing each of these paradigms. In this context, and for the MapReduce framework, we are inspired from (Palla, 2009) which provides an insight into the Map/Reduce framework in terms of Input/Output cost, presents a thorough analysis of the map and reduce tasks and introduces a cost model for each one. Exploiting the open source Hadoop implementation of the MapReduce framework, having the partitioning over the key space and the grouping based on the same key as main components of the standard MapReduce data flow, this model develops a theoretical cost model to evaluate the Input/Output cost induced on each node during the execution of a task. It studied the data-flow of a map and reduce task and extracted a cost model that evaluates the Input/Output cost related to each task. Based on this model, we used it to develop our theoretical cost model in order to evaluate and compute the total cost of each job for a MapReduce process. We implement a MapReduce cost model, and we propose a refinement.

Performance cost model is based on its dependence on various factors. Thus, in terms of parameters, the MapReduce cost model also depends on the input file to be processed. A MapReduce process takes an input file. This file is based on many parameters: the first is the file type , and then we focus on the ability of the framework to handle multiple file types compared with traditional DBMSs which handles only tables; Then, the file size; the number of records in the file and the number of attributes per record; The origin of the input file that can be given or resulting from other operations. Thus, from the standpoint of system the file is implemented as a table, hence the presence of a Row Identifier (RID). For physical parameters, it includes the number of nodes forming process execution cluster, the number of map and reduce tasks that can run simultaneously on a single node, the cost of data transfer and the size of the Row Identifier.

3.4.1 Input/Output cost model assumptions

The proposal of a mathematical model of cost is generally based on simplifying assumptions to simplify the development of mathematical functions of different query execution costs. In our study we consider the following assumptions:

- The data are distributed uniformly and attributes are independent; these both assumptions are widely used in estimating execution time queries in DBMSs.
- The CPU cost is negligible compared to the Input/Output cost.
- Join strategy is Repartition join.
- All nodes have the same physical characteristics and the buffer size of each node is large enough to hold the data.

Based on these assumptions we will detail in the following sections the Input/Output cost analysis of the two main phases of a MapReduce process.

3.4.2 Input/Output cost analysis of the Map phase

An input file in a MapReduce process will be split into one or more blocks in a size already predefined and configurable. The configuration of this parameter depends on the size of the input, and depends on the performance of resources allocated. This predefined size of a block is a performance criterion since it influences the number of Map tasks, the time data loading and the execution time of tasks. Then, each block is replicated across several machines to prevent data loss is case of a single machine failure. The number of map task to perform will be equal to the number of blocks. The figure 3.2 shows the steps involved in performing a map task during a MapReduce process.

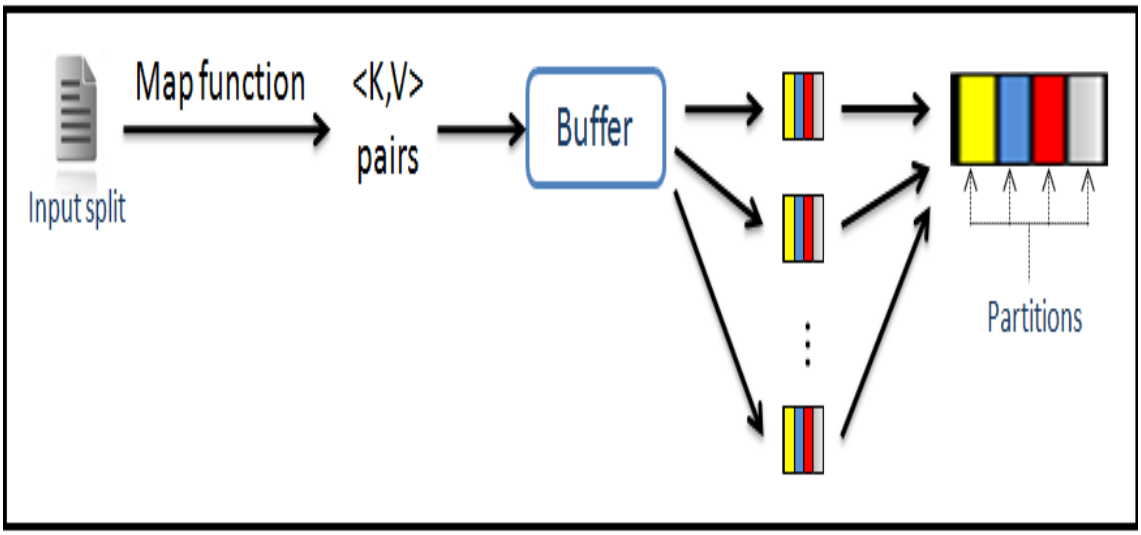

Figure 3.2: Map phase data flow with a single map task

For each map task, the triggered map process read the input and writes the relevant output to disk. In addition, the process includes sorting and partitioning stages where writes to buffer and spills to disk take place.

The performance of the tasks map as well as the estimation of the cost of execution of every task depends of a set of configurable parameters related to the platform. The table 3.2 outlines the main parameters involved in the cost model for the Map task.

Table 3.2: Platform main parameters for Map task

Parameter	Description
Dfs.blocksize	Default block size.
Io.sort.mb	The total amount of buffer memory to use while sorting filesss.
Io.sort.record.percent	The percentage of io.sort.mb dedicated to tracking record boundaries.
Io.sort.spill.percent	The soft limit in either the buffer or record collection buffers.
Io.file.buffer.size	The size of buffer for use in sequence files. (should sbe a multiple of hardware page size). It determines how much data is buffered during read and write operations.
Io.sort.factor	The number of streams to merge at once while sorting files. This determines the number of open file handles.

Each map task can be divided into three phases: reading phase, buffering phase and the writing phase.

3.4.2.1 Reading phase

During reading phase, each map task reads the input which is called split. The Master, using knowledge of the file system, tries to assign the processing of each block to the Worker on which the data block is actually stored. Each mapper reads the input record by record and converts it into a Key/Value pairs. Based on the user-defined map function, the propagated key/Value pairs are transformed on a new set of Key/Value pairs, which is pushed to the subsequent phase. Since the input is directly fetched from the HDFS, the Input/Output cost of this phase equals to zero.

3.4.2.2 Buffering phase

During buffering phase, the map output (set of Key/Value pairs) is serialized and written to a circular buffer. Thus, three procedures take place: partitioning, sorting and spilling to disk.

The configuration property "io.sort.mb" defines the available buffer size. In details, there is a main buffer called serialization buffer which is dedicated to collect the output; and the accounting buffers: two additional buffers which maintain location of key and value for records in order to facilitate the sorting procedure. The value of the configuration property "Io.sort.record.percent" defines the percentage of the buffer size dedicated to storing meta-data:

- Meta-data buffer size:

$$buff_{mtdt} = (io.sort.mb) * (io.sort.record.percent) \tag{3.4.1}$$

- Serialization buffer size:

$$buff_{serial} = (io.sort.mb) - [(io.sort.mb) * (io.sort.record.percent)] \tag{3.4.2}$$

Each emitted record is serialized into the main buffer and meta-data are stored into accounting buffers. Since the configuration property "Io.file.buffer.size" determines the amount of data buffered during read and write operations, the maximum number of records the buffer can store will be equal to available buffer size divided by the size of records which can be obtained by dividing the split size by the number of records contained in each block of input data. When either of the serialization or the meta-data buffer exceeds a threshold, the contents of the buffers will be sorted and spilled to disk in the background. The spilling threshold is triggered according to the percentage defined by the property configuration "Io.sort.spill.percent" which determines the soft limit in either of the serialization and meta-data buffers. In most cases, the "Io.sort.record.percent" that determines the size of the meta-data buffer, is chosen low enough (0.05 by default); Consequently the spill threshold of the accounting buffers is the first one to be reached. Once reached, a thread will begin to spill the contents to disk in the background.

The Input/Output cost of the buffering phase equals to the cost of writing the whole input split, divided in spills.

3.4.2.3 Writing phase

This last phase for the map task requires that the spill files are merged in order to end up with a single sorted output file. To achieve this, a merging algorithm is implemented and is repeated as many times as the number of the partitions determined by user. The spill files consist the input of this phase which are stored on the local disk. Each spill file is divided in sorted partitions, then each iteration of the algorithm merges the same partitions of each spilled file. The figure shows an example of running the merging algorithm.

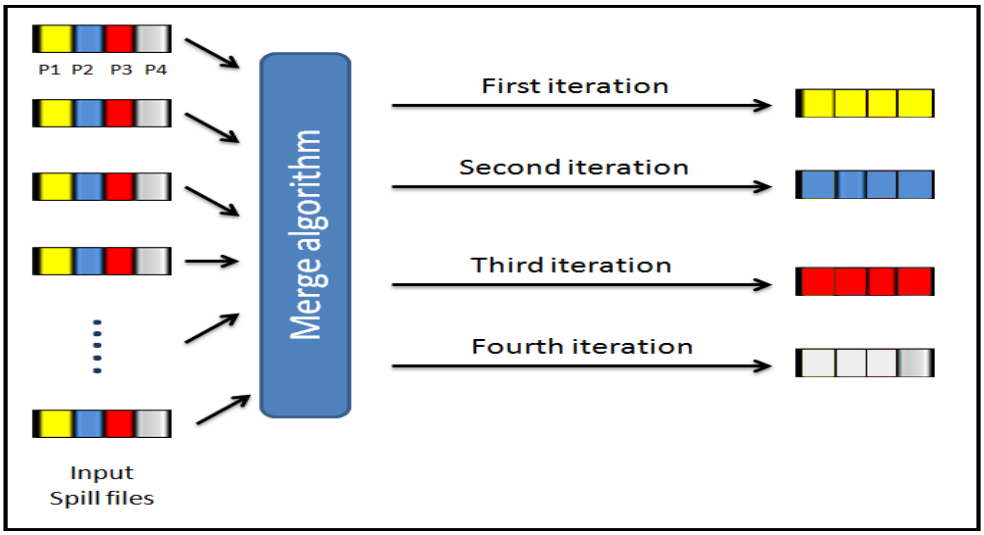

Figure 3.3: Map phase data flow with a single map task

That is, as illustrated in the example presented in the figure 3.3, for 4 sorted partitions in each spill file,P1 P2 P3 and P4 the merging algorithm will be repeated 4 times and during each iteration the spill files contribute the same partition; the first iteration will merge the P1 partitions of each spill file, the second will merge

the P2 partitions, the third for merging the P3 partitions and the fourth for the P4 partitions. The set of the same partitions being merged during each iteration is called set of segments where each segment is a set of records that belong to the same partition and to the same spill file.

3.4.2.4 Map task total cost

The total cost of the map task includes the cost of reading the whole input split, the cost of writing the spill files and finally the cost of merging. It is the sum of the costs induced by the reading, buffering and writing phases.

$$Cost_{Map} = Cost_{read} + Cost_{buff} + Cost_{write} \tag{3.4.3}$$

3.4.3 Input/Output cost analysis of the Reduce phase

The high degree of parallelization and the overlap in the operations of the reduce task have made the complication of the reduce phase. The reduce task includes three phases: the shuffle/copy, sort and reduce phases which are depicted in the figure 3.3.

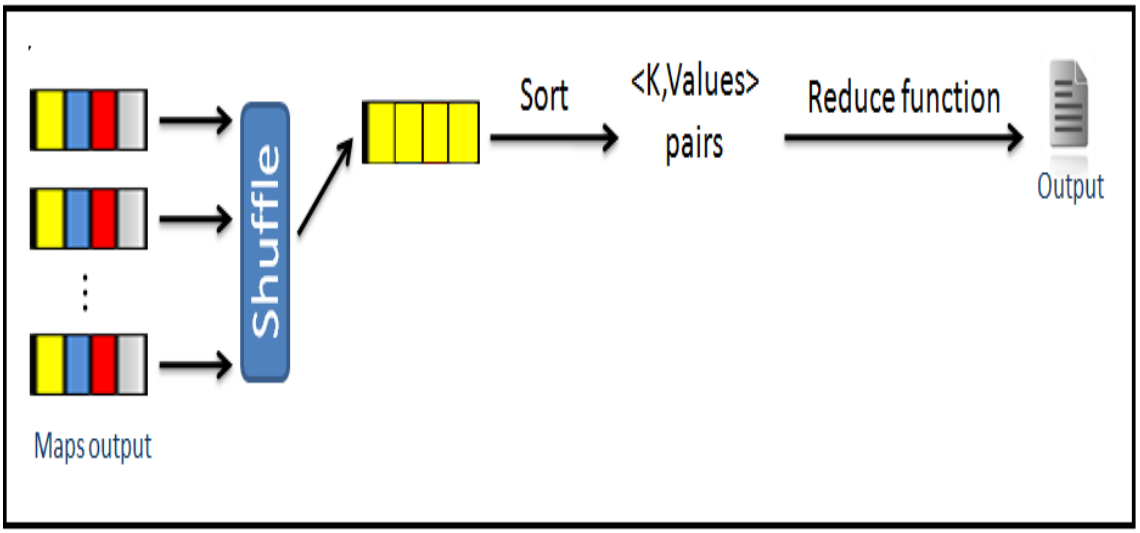

Figure 3.4: Reduce phase data flow with a single reduce task

Same as every part of the MapReduce process, the reduce task and mainly its Input/Output cost, highly depends on the configuration properties that regulate the setup of the cluster. The table 3.3 outlines the main parameters involved in the cost model for the Reduce task.

Table 3.3: Platform main parameters for Reduce task

Parameter	Description
Mapred.child.java.opts	Heap size associated with each task.
Mapred.job.shuffle.input.buffer.percent	Percentage of memory to be allocated from the maximum heap size to sorting map outputs during the shuffle.
Mapred.job.shuffle.merge.percent	Percentage of the total memory allocated to sorting in memory map outputs.
Io.sort.factor	The number of streams to merge at once while sorting files. This determines the number of open file handles.

In every map-reduce job, there would be as many reduce tasks as the number of partitions, and the Map output files actually constitute the input of the reducers.

For simplicity, we still assume that all map outputs received by the reducer are of approximately equal size.

3.4.3.1 Shuffle/Copy phase

During the copy phase each reduce task copies the map outputs locally. The property "mapred.child.java.opts" provides a memory of a prespecified size to each task. However, only a percentage "mapred.job.shuffle.input.buffer.percent" of it can be used for copying.

$$MaxMemory_{shuff} = (mapred.child.opts) * (mapred.job.shuffle.input.buffer.percent) \tag{3.4.4}$$

A size of less than 25% of the $\text{MaxMemory}_{\text{shuff}}$ will allow map output to be written in memory, otherwise it is propagated to disk. When the in-memory buffer reaches a threshold size, the in-memory outputs are merged and spilled to disk creating a new local file. this threshold is determined by the configuration property "mapred.job.shuffle.merge.percent".

$$InMemory_{threshold} = (MaxMemory_{shuff}) * (mapred.job.shuffle.merge.percent) \tag{3.4.5}$$

3.4.3.2 Sort phase

This second phase of the reduce task starts after all the map outputs have been successful copied in memory and/or on disk. This phase is carried out in a way that maintains the Map phase sort order.

3.4.3.3 Reduce phase

During this last phase of the reduce task, the reduce function is invoked for every pair of key/value. The output of this phase is written directly to the output file system in the HDFS. As no bytes are read or written locally, then it results in no Input/Output cost.

3.4.3.4 Reduce task total cost

The total cost of the reduce task, is the following sum of the three involved phases: copy/shuffle phase, sort phase and reduce phase.

$$Cost_{Red} = Cost_{Shuff} + Cost_{sort} + Cost_{red} \tag{3.4.6}$$

3.5 Conclusion

In this chapter, we presented briefly a simplified overview of a MapReduce Input/Output cost model in order to use it for the implementation of our proposed approach.

In the next chapters, we detail this approach which propose to integrate MapReduce paradigm with an RDBMS and we present the results of our work evaluation.

Chapter 4

The proposed approah: OLAP over Big Data

4.1 Introduction

Data processing needs are changing with the ever increasing amounts of both structured and unstructured data. While the processing of structured data typically relies on the well developed field of relational database management systems (RDBMSs), MapReduce is a programming model developed to cope with processing immense amounts of unstructured data.

In this context, we present in this chapter our approach for integrating the two paradigms RDBMS and MapReduce with the aim of minimizing the Input/Output cost. First, we present the general context and our contribution, then we detail our approach implementation.

4.2 Context and contribution

Businesses are collecting more information than ever, and the amount of data being kept is increasing dramatically. It has become essential to have powerful tools to verify and analyze information in order to support the decision-making process and to make the decision the most adapted at a given moment.In this context, Business

Intelligence (BI) is installed and the new methods such as OLAP and data warehousing are introduced. Thus, data warehouses, as presented in (Furtado, 2009), are the basis of decision-support (OLAP OnLine Analytical Processing), which requires complex decision-support queries and very time-consuming process because these queries require star joins between the fact table and the dimension tables. Several solutions are possible and used to improve the performance of these applications.

Nowadays, the database market has received much attention and is increasingly growing. As proposed in (Abouzeid *et al.*, 2009), along with evolution data bases world, increasing volume of data and the need of data management, there are two popular schools of thought for performing large-scale data processing that does not fit into memory. The one is classic DBMSs that are capable of efficiently managing, updating and querying tables and structured data. The other school of thought suggests analytical function into NoSQL Database management system, presented in (Strauch *et al.*, 2011), that appoints a category of DBMSs based on non-relational distributing architecture. For the first school of thought, we will look to ORDBMS that present the result of merge of RDBMS and ODBMS; and specifically PostgreSQL, presented in (Worsley and Drake, 2002), the most notable research project in the field of ORDBMSs. For the second school, we will be interested in Hadoop MapReduce, proposed in (Dean and Ghemawat, 2008), the simple programming framework popularized by Google but yet powerful way to implement distributed applications without having deeper knowledge of parallel programming. Thus, the administrator must choose the optimal or near-optimal solution. The choice of the optimal configuration of structures is based on the evaluation of the quality of configurations generated, which can be done using cost models. In this context, we propose, a database-Hadoop hybrid approach. We will discuss Hadoop MapReduce and RDBMS as competing and complimentary paradigms in order to benefit of both approaches and overcome their limitations. The basic idea behind our approach is

based on the cost model to approve execution and selectivity of solutions based on the estimated cost of execution. To support the decision making process for analyzing data while minimizing costs, we proposed to compare the estimates of the costs of running a query on Hadoop MapReduce compared to PostgreSQL to choose the least costly technology. For a better control, we will proceed to a thorough query analysis.

4.3 Pushed analysis of a query's execution

In this section we will detail the process of our proposed approach. To have a better explanation, quite the stages of the process of the proposed approach will be illustrated by examples of processing of the following OLAP query.

```
select d_year, s_nation, p_category,
       avg(lo_revenue-lo_supplycost) as avg_profit
from lineorder, dates, customer, supplier, part
where lo_custkey = c_custkey
and lo_suppkey = s_suppkey
and lo_partkey = p_partkey
and lo_orderdate = d_datekey
and c_region = 'AMERICA'
and s_region = 'AMERICA'
and (d_year = 1997 or d_year = 1998)
and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
group by d_year, s_nation, p_category
order by d_year, s_nation, p_category
```

The detailed analysis of the queries execution costs showed a gap mattering between both paradigms. Hence the idea of the thorough analysis of the execution process of each query and the implied cost. To better control the cost difference between costs of Hadoop MapReduce versus PostgreSQL on each step of the query's execution process, we propose to dissect each query for a set of operations that demonstrates the process of executing the query. In this way we can check the impact of the execution of each operation of a query on the overall cost and we can control the total cost of the query by controlling the partial cost of each operation.

In this context, we are inspired from a work done in (Boukorca *et al.*, 2014) to provide a detailed execution plan for OLAP queries. This execution plan zooms in on the sequence of steps of the process of executing a query. It allows detailing the various operations of the process highlighting the order of succession and dependence. The presented execution plan below illustrates various execution phase of the proposed sample query.

```
2742857     Selection Operation :(24) :  with predecessor=23

2742857     Join Operation :(23) :  with left=22 and right=15

13696000    Join Operation :(22) :  with left=21 and right=18

34285714    Join Operation :(21) :  with left=20 and right=16

171360930   Join Operation :(20) :  with left=0 and right=17

600000000   Access to Base Table :LINEORDER(0)

730         Selection Operation :D_YEAR=1997 OR D_YEAR=1998(17) :
            with predecessor=1

2556        Access to Base Table :DATES(1)

40000       Selection Operation :S_REGION='AMERICA'(16) :
            with predecessor=3

200000      Access to Base Table :SUPPLIER(3)

560000      Selection Operation :P_MFGR='MFGR#1' OR
            P_MFGR='MFGR#2'(18) :  with predecessor=2

1400000     Access to Base Table :PART(2)

600000      Selection Operation :C_REGION='AMERICA'(15) :
            with predecessor=4

3000000     Access to Base Table :CUSTOMER(4)
```

In addition to dissect the implementation process, the execution plan details for each operation the amount of data by the accuracy of the number of records involved and the dependence implemented in the succession of phases. These parameters will be needed to calculate the cost involved in each operation. After distinguishing the different operations of the query, the next step is to calculate the unit cost of each operation. As part of our approach we will learn on this work in order to dissect each query and focus on each separate operation. That way we can control the different stages of the execution process of each query with the aim of calculate the cost implied by each operation as well as its influence on the total cost of the query. Therefore we can control the cost of each query to support the decision making process and the selectivity of the proposed solutions based on the criterion of cost minimization.

4.4 Queries cost estimation

Having identified all operations performed during the query execution process the next step is then to calculate the cost implied in each operation independently, in both paradigms PostreSQL and Hadoop MapReduce with the aim of controlling the estimated costs difference according to the operations as well as the total cost of query execution. At this stage we consider each operation independently to calculate an estimate of its cost execution on PostreSQL on one hand then on MapReduce on the other hand.

4.4.1 Cost estimation on PostgreSQL

PostgreSQL is a free object-relational database management system. It runs on various hardware platforms and under different operating systems and it is widely recognized for its stable behavior, and for its extensive programming possibilities via PL/pgSQL. PostgreSQl provides the possibility of itemize each operation by an incremental value of the Input/Output cost implied in each step. The figure below shows the detailed execution process of the query example, proposed on the previous section, on PostgreSQL:

```
Sort  (cost=200309.67..200309.89 rows=90 width=28)
Sort Key: dates.d_year, supplier.s_nation, part.p_category
->  HashAggregate  (cost=200305.62..200306.75 rows=90 width=28)
->  Hash Join  (cost=8402.01..200036.44 rows=21535 width=28)
Hash Cond: (lineorder.lo_partkey = part.p_partkey)
->  Hash Join  (cost=1084.42..190998.14 rows=59913 width=24)
Hash Cond: (lineorder.lo_custkey = customer.c_custkey)
->  Hash Join  (cost=147.51..186462.47 rows=299964 width=28)
Hash Cond: (lineorder.lo_orderdate = dates.d_datekey)
->  Hash Join  (cost=57.73..179119.85 rows=1134183 width=28)
Hash Cond: (lineorder.lo_suppkey = supplier.s_suppkey)
->  Seq Scan on lineorder  (cost=0.00..145216.67 rows=6000967 width=24)
->  Hash  (cost=53.00..53.00 rows=378 width=12)
->  Seq Scan on supplier  (cost=0.00..53.00 rows=378 width=12)
Filter: ((s_region)::text = 'AMERICA'::text)
->  Hash  (cost=81.34..81.34 rows=676 width=8)
->  Seq Scan on dates  (cost=0.00..81.34 rows=676 width=8)
Filter: ((d_year = 1997) OR (d_year = 1998))
->  Hash  (cost=862.00..862.00 rows=5992 width=4)
->  Seq Scan on customer  (cost=0.00..862.00 rows=5992 width=4)
Filter: ((c_region)::text = 'AMERICA'::text)
->  Hash  (cost=6067.00..6067.00 rows=71888 width=12)
->  Seq Scan on part  (cost=0.00..6067.00 rows=71888 width=12)
Filter: (((p_mfgr)::text = 'MFGR#1'::text) OR ((p_mfgr)::text = 'MFGR#2'::text))
```

In PostgreSQl platform, the command "explain" show the execution plan of a statement. This command displays the execution plan that the PostgreSQL planner generates for the supplied statement. Besides the succession of the executed operations, the most critical part of the display is the estimated statement execution cost, which is the planner's guess at how long it will take to run the statement (measured in cost units that are arbitrary, but conventionally mean disk page fetches). It include information on the estimated start-up and total cost of each plan node, as well as the estimated number of rows. Actually two numbers are shown: the start-up cost before the first row can be returned, and the total cost to return all the rows. Therefore for each operation, the cost should be the difference between these two

values.

4.4.2 Cost estimation on Hadoop MapReduce

In a MapReduce system, a query star join between F (fact table) and n dimension tables, runs a number of phases, each phase corresponds to a MapReduce job. So, for MapReduce paradigm we have first to extract the number of jobs that will be run for executing the query. The MapReduce job number depends on the number of joint and the presence or absence of aggregation and sorting data. There are three cases of figure: The request contains only n successive joint operations between F and n dimension tables; Join operations are followed by a process of grouping and aggregation on the results of the joint; Sort is applied to the results. In this context, we propose to relie on the equation (4.4.1) presented bellow, and inspired from (Brighen, 2012). This equation allows to determine the number of MapReduce jobs implied in the execution of a given OLAP query. This number can be estimated by the following formula:

$$Nbr_{job}(q) = n + x \tag{4.4.1}$$

The "n" refers to the number of dimension tables. The "x" can be equal to: 0, if the query involves only join operations; 1, if the query contains grouping operations and aggregation; 2, if the results are sorted. After identifying all the jobs of query execution, the next step is to calculate the Input/Output cost implicated in each job. In this stage, we relied on the mentioned work done by Ahcene Bokorca to extract the amount of data and the number of records involved in each operation. These two parameters will be used in the refined cost model presented in the previous chapter,

in order to calculate the Input/Output cost of each job.

4.5 Discussion

The analysis of the results of the estimated costs independently for each operation showed the high cost of the first join operation executed on PostgreSQL, and a noticeable difference for the Hadoop MapReduce paradigm. This can be explained by the fact that in the case of data warehouses, the fact table is still the largest table in terms of number of tuples, which explains the high cost of its analysis. Hence our idea of performing the first joint operation that integrates fact table on Hadoop MapReduce framework which proves competence for heavy processing to be performed on a large volume of data. In this way we try to minimize the cost of the query execution by minimizing the cost of the most expensive operation. Other operations required by the query such as aggregation, sorting, in addition to other join operations will be passed on PostgreSQL.

On the other hand, nowadays organizations are increasingly concerned about data. This can be explained by the strategic role of data management for the enterprise.This same data contain very often confidential and exclusive information such as information relative to the customers or to the financial results, which explain the importance of learning the best practice in quality control of the data and ensure its privacy and security, especially in the era of cloud computing presented in (Vouk, 2008) and the remote storage and data analysis. Therefore we try to limit the external data analysis in order to preserve the privacy of the data especially when it comes to the dimension tables that contain data values from the fact table that contains only foreign keys. Therefore, by moving only the first join operation on Hadoop MapReduce we try to limit the external data analysis in order to preserve the data privacy especially when it comes to the dimension tables that contain data

values versus the fact table that contains only foreign keys.

4.6 Conclusion

We detailed in this chapter the principles of our proposed approach: OLAP over Big Data.

In order to evaluate the performance of our method, the next chapter will describe the implementation of our system and the results generated by the framework besides the evaluation of its effectiveness.

Chapter 5

Experiments and results

5.1 Introduction

In order to validate our approach presented in the previous chapter and prove its performance, we present in this section the results of experiments conducted to evaluate the proposed approach performance as well as gain cost compared with the cost required by each platform independently. This chapter presents the work environment and used compares the results obtained by the approach.

5.2 Experimental environment

The experiments involve two DBMSs: an ORBMS PostgreSQL and a NoSQL DBMS Hadoop MapReduce. To test and compare our theoretical expectations with the real values, we set up a cluster consisting of one node and conducted a series of experiments. For all the experiments, we use the version 9.3 of PostgreSQL. For Hadoop, we used the version 2.0.0 with a single node as worker node hosting TaskTracker and DataNode, and as the master node hosting JobTracker and NameNode. The tables 5.1 and 5.2 present the platform parameters for Hadoop MapReduce framework for both phases Map and Reduce.

Table 5.1: Platform parameters for the Map task

Parameter	Value
Dfs.blocksize	128 MB
Io.sort.mb	50
Io.sort.record.percent	0.05
Io.sort.spill.percent	0.80
Io.file.buffer.size	65536
Io.sort.factor	64

Table 5.2: Platform parameters for the Reduce task

Parameter	Value
Mapred.child.java.opts	145171557 MB
Mapred.job.shuffle.input.buffer.percent	0.70
Mapred.job.shuffle.merge.percent	0.66
Io.sort.factor	64

In addition, we implemented two jobs: the first (job 1) get the occurrence of a record, and the second (job 2) search the not null rows. Then, we run each job on two text files of different size and different records number (presented in the table 5.3 below).

Table 5.3: Experimentation files

	Size	Records	Attributes
File 1	262 MB	36013069	1
File 2	228 MB	8946601	4

We worked on a workload of 30 queries OLAP (presented in the annexe). The training data consisted of a data warehouse of 100GB of data with a fact table (Lineorder) and 4 dimension tables (Dates, Part, Supplier, Customer) whose the record's number per table and the instance size are shown in the table 5.4 below.

Table 5.4: Data warehouse's caracteristics

Table	Record's number	Size of an instance
Lineorder	600000000	84
Dates	2556	92
Part	1400000	84
Supplier	200000	85
Customer	3000000	95

Figure 5.1 details the attributes of the data warehouse's fact table. It shows the table's attributes list and specify the type and constraints of each attribute.

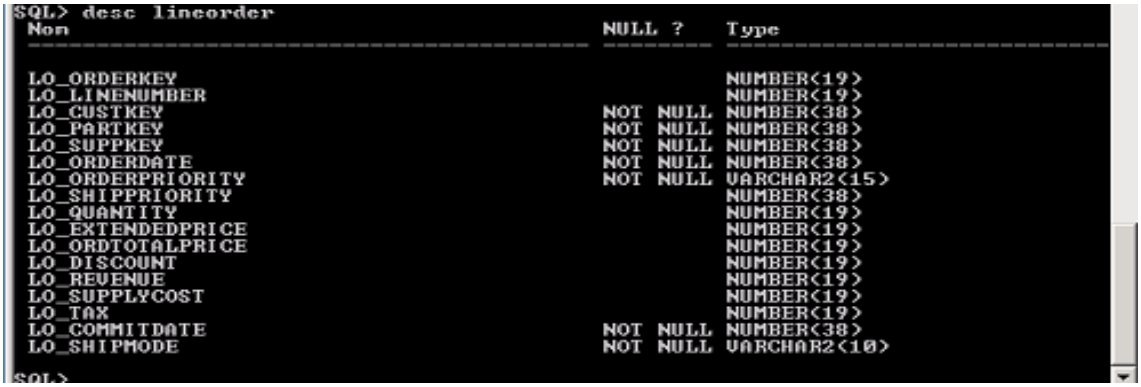

Figure 5.1: Fact table: Lineorder

Figures 5.2, 5.3, 5.4 and 5.5 detail the attributes of the data warehouse's dimension tables respectively Dates, Part, Supplier and Customer. Each figure details the attributes of the corresponding table, and specify for each table the type and constraints of each attribute contained in the attributes list.

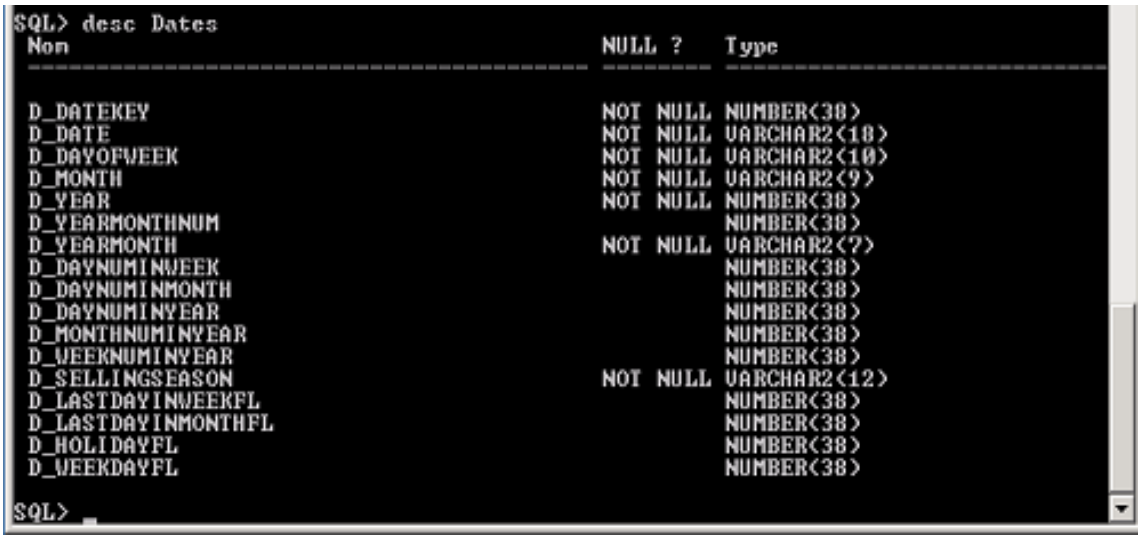

Figure 5.2: Dimension table: Dates

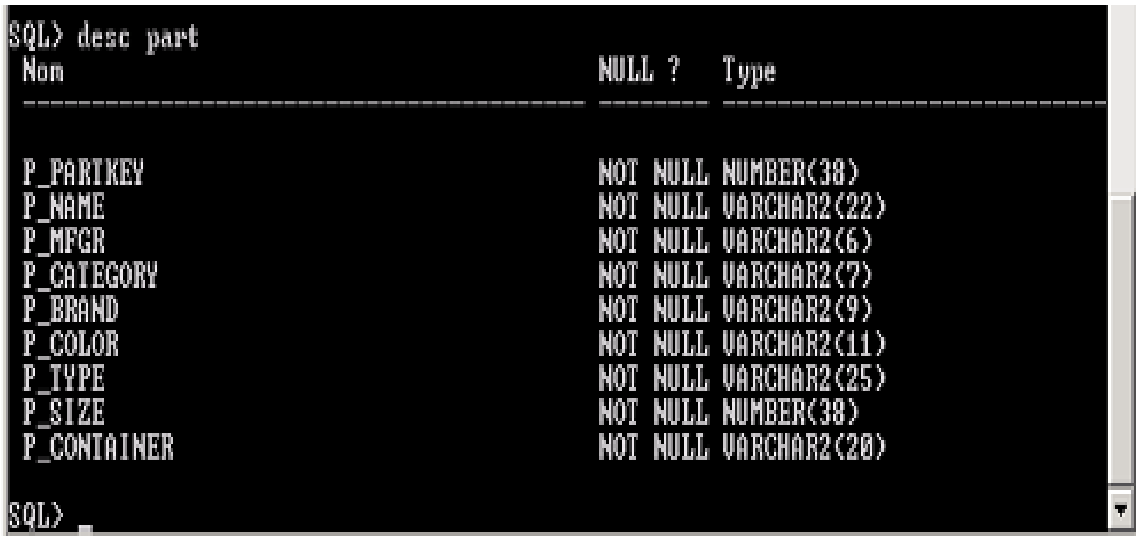

Figure 5.3: Dimension table: Part

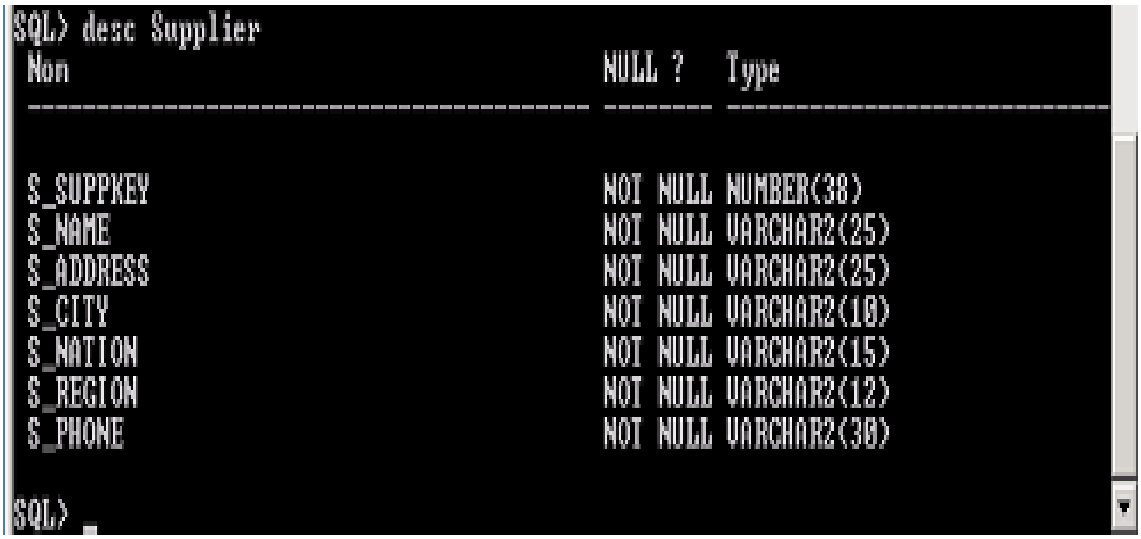

Figure 5.4: Dimension table: Supplier

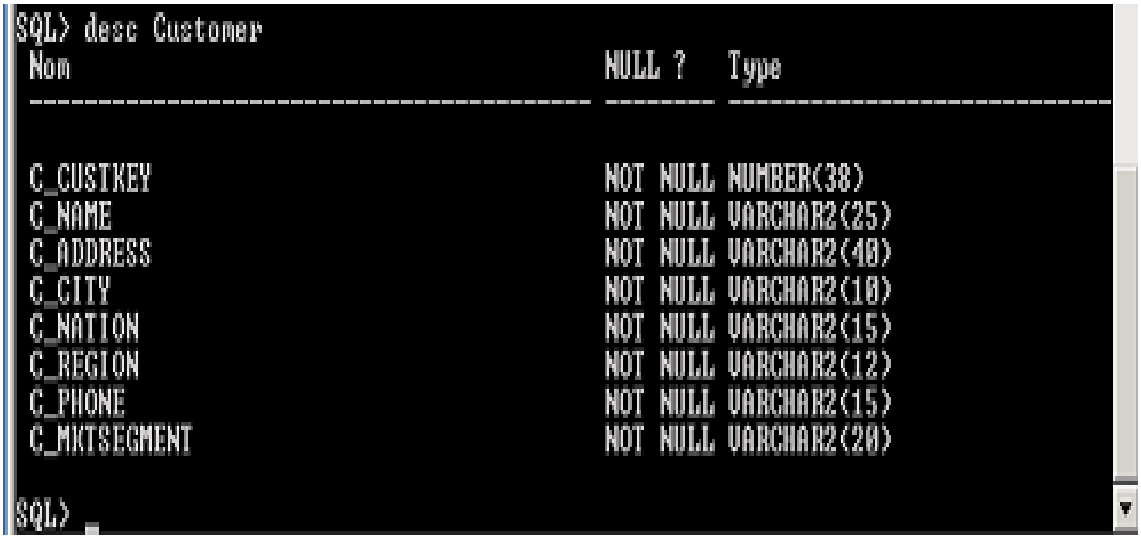

Figure 5.5: Dimension table: Customer

After presenting the work environment, we will present and discuss the obtained results.

5.3 Results presentation

Our approach proposes an hybrid model between ORDBMS and Hadoop MapReduce, based on the comparison of Input/Output costs on the both paradigms. At this stage, we will present the results of MapReduce refined Input/Output cost model and the results of the implementation of the proposed approach.

5.3.1 MapReduce refined Input/Output Cost model results

The results returned by our MapReduce refined Input/Output cost model vary of 87-90% compared to the actual values of running jobs on MapReduce. The table 5.5 presents the results returned by the refined Input/Output cost model for running two implemented jobs on both files described in the previous section (Theoretical cost), well as the real Input/Output costs of the same process on the Hadoop MapReduce platform (MR cost).

Table 5.5: Input/Output cost model results

Job	File	Theoretical cost	MR cost	%
Job 1	File 1	2594790162	2320781012	89%
	File 2	621699124	539670838	87%
Job 2	File 1	2594790162	2350781018	90%
	File 2	621699124	539670844	87%

The theoretical cost is the same for the execution of both jobs on the first file, and even for the execution of both jobs on the second file. This can be explained by the fact that our refined Input/Output cost model take in parameters the file size and the number of records, but it is not sensitive to the nature of treatment implemented in the job to run. We draw interesting conclusions by comparing the predicted values with the real ones provided by the framework. Observing the values in table 5.5, we conclude that our refined cost model provided a good approximation of the real

Hadoop MapReduce Input/Output cost. The estimated costs are only slightly over predicted compared to the real values. Theoretical costs are from 87 to 90% of the real Hadoop MapReduce costs.

We will use this Input/Output cost model to the implementation of our proposed approach.

5.3.2 Proposed approach results

The results of the application of the proposed approach are presented in the table 5.6 below. It shows the total Input/Output cost of running the workload on Hadoop MapReduce (Tot_cost_MR), the total Input/Output cost of running the workload on ORDBMS PostgreSQL (Tot_cost_PG), the total Input/Output cost of the workload by applying our proposed approach (O.O.B.G), the total Input/Output cost of the workload by choosing for each operation of each query the lowest cost between Hadoop MapReduce and PostgreSQL (Best_Cost_MR_PG).

Table 5.6: proposed approach result

Tot_cost_MR	Tot_cost_PG	O.O.B.D	Best_Cost_MR_PG
505579305,9	786297800,8	369473436,7	365360413

Observing the values presented in table 5.6 illustrate the difference of the cost saved up by the application of the proposed approach. The total cost of running all the workload under the proposed approach is only slightly over predicted compared to the cost estimated cost to run the workload by choosing for each operation of each query the lowest cost between the two proposed paradigm (Hadoop MapReduce and PostgreSQL).

The histogram presented in the figure 5.1 shows the gain of the Input/Output cost by running the workload independently on Hadoop MapReduce and on postgreSQL.

In addition, it compare the Input/Output cost of the workload obtained by applying our proposed approach to the Input/Output the cost obtained if we choose the lower cost for each operation contained in the execution plan of each query.

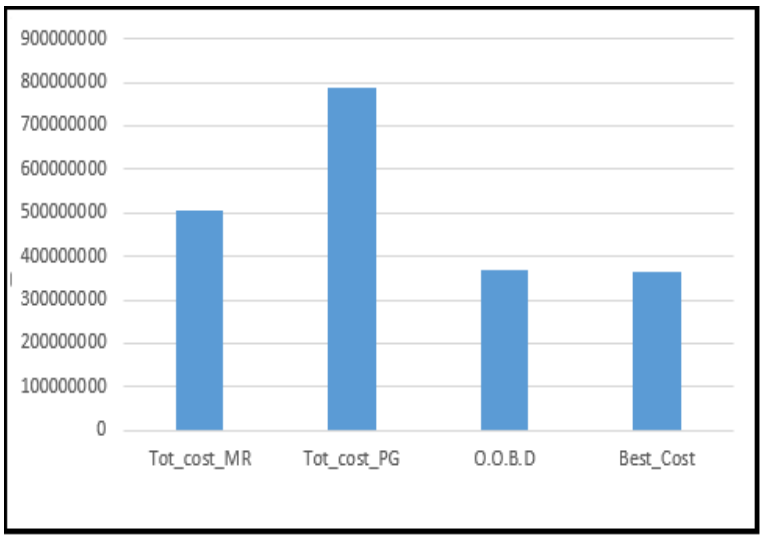

Figure 5.6: Comparative results

The figure 5.1 illustrates the notable difference of the Input/Output cost of running the workload by applying the proposed approach compared to the Input/Output running cost on the two paradigms PostgreSQL and Hadoop MapReduce separately. The gain estimated of the Input/Output cost saved up by running the workload tanks to applying the proposed approach is of 27% compared to Hadoop MapReduce and 53% compared to PostgreSQL. This percentage gain proves the performance of our proposed approach. In addition, the cost returned by our proposed approach is 98.8% of the optimal cost obtained in the case of choosing for each operation of each query the lowest cost between Hadoop MapReduce and PostgreSQL. Also, we have to highlight the advantage of our proposed approach in the limitation of the external data analysis in order to preserve its privacy, by moving only the first join operation on Hadoop MapReduce. This way we try to preserve the data privacy contained in

the dimension tables that contain data values versus the fact table that contains only foreign keys.

5.4 Conclusion

We exposed in this chapter our system implementation and its evaluation. The comparison of our system with results of each paradigm separately proved the higher performance of our search results confirming our model's effectiveness.

Conclusion

Given the exploding data problem, the world of databases has evolved which aimed to escape the limitations of data processing and analysis. There has been a significant amount of work during the last two decades related to the needs of new supporting technology for data processing challenged by the rise of data generation to escape the limitations of existing technologies for data analysis.

In order to achieve the goal of this, we went through several stages. We began by gathering information and reading articles and student theses. Then we studied the two approaches proposed by the research community and known to be good methods for data process and analysis which are DBMSs and precisely the ORDBMSs (PostgreSQL), and NoSQL DBMSs and precisely Hadoop MapReduce.

In this context, we propose a new approach to optimizing Input/Output cost. At first, we introduce some notions about DBMSs and NoSQL. Then we dealt with the recently introduced Google's programming model Map/Reduce and we presented the cost model on which we based on implementation of our proposed approach of a database-Hadoop hybrid model. We then presented the results of our experiments.

Our experiments show that our proposed approach is able to approach the performance of MapReduce and PostgreSQL independently. The results revealed that our proposed techniques outperform the PostgreSQL DBMS by up to 40%, and the Hadoop MapReduce by up to 70%, compared to the Input/Output cost of OLAP workload running.

Having discussed the conclusions drawn by our work, our attention can turn to what else may be done in the future. Interesting perspectives emerge to further strengthen the proposed approach. One interesting direction would be that of a systems-level hybrid.

Appendix:

workload used for approach evaluation

Q1: select sum(lo_extendedprice*lo_discount) as revenue from lineorder, dates where lo_orderdate = d_datekey and d_year = 1993 and lo_discount >= 1 and lo_discount <= 3 and lo_quantity < 25

Q2: select count(*) from lineorder, dates where lo_orderdate = d_datekey and d_year = 1993 and lo_discount >= 1 and lo_discount <= 3 and lo_quantity < 25

Q3: select sum(lo_extendedprice*lo_discount) as revenue from lineorder, dates where lo_orderdate = d_datekey and d_year = 1993

Q4: select count(*) from lineorder, dates where lo_orderdate = d_datekey and d_year = 1993

Q5: select sum(lo_revenue), d_year from lineorder, dates, part, supplier where lo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#2221' and s_region = 'ASIA' group by d_year order by d_year

Q6: select sum(lo_revenue) from lineorder, part where lo_partkey = p_partkey and p_brand = 'MFGR#2221'

Q7: select avg(lo_revenue), d_year from lineorder, dates, part, supplier where lo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#2221' and s_region = 'ASIA' group by d_year order by d_year

Q8: select sum(lo_revenue), d_year from lineorder, dates, part, supplier where lo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey

and p_brand = 'MFGR#2221' and s_region = 'EUROPE' group by d_year, p_brand order by d_year, p_brand

Q9: select sum(lo_revenue) from lineorder, part, supplier where lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#2221' and s_region = 'EUROPE'

Q10: select count(*), d_year from lineorder, dates, part, supplier where lo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#2221' and s_region = 'EUROPE' group by d_year order by d_year

Q11: select c_nation, s_nation, d_year, sum(lo_revenue) as revenue from lineorder,customer, supplier, dates where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and c_region = 'ASIA' and s_region = 'ASIA' and d_year >= 1992 and d_year <= 1997 group by c_nation, s_nation, d_year order by d_year asc, revenue desc

Q12: select s_nation, sum(lo_revenue) as revenue from lineorder, supplier where lo_suppkey = s_suppkey and s_region = 'ASIA' group by s_nation order by revenue desc

Q13: select s_nation, count(*) as revenue from lineorder, supplier where lo_suppkey = s_suppkey and s_region = 'ASIA' group by s_nation order by revenue desc

Q14: select s_nation, d_year, sum(lo_revenue) as revenue from lineorder, supplier, dates where lo_suppkey = s_suppkey and lo_orderdate = d_datekey and s_region = 'ASIA' and d_year >= 1992 and d_year <= 1997 group by s_nation, d_year order by d_year asc, revenue desc

Q15: select c_nation, s_nation, d_year, avg(lo_revenue) as avg_revenue from lineorder,customer, supplier, dates where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and c_region = 'ASIA' and s_region = 'ASIA' and d_year >= 1992 and d_year <= 1997 group by c_nation, s_nation, d_year order by d_year asc, avg_revenue desc

Q16: select c_nation, s_nation, d_year, count(*) from lineorder,customer, supplier, dates where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate

= d_datekey and c_region = 'ASIA' and s_region = 'ASIA' and d_year >= 1992 and d_year <= 1997 group by c_nation, s_nation, d_year order by d_year asc

Q17: select c_city, s_city, d_year, sum(lo_revenue) as revenue from lineorder,customer, supplier, dates where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and c_nation = 'UNITED STATES' and s_nation = 'UNITED STATES' and d_year >= 1992 and d_year <= 1997 group by c_city, s_city, d_year order by d_year asc, revenue desc

Q18: select s_city, sum(lo_revenue) as revenue from lineorder, supplier where lo_suppkey = s_suppkey and s_nation = 'UNITED STATES' group by s_city order by revenue desc

Q19: select s_city, avg(lo_revenue) as avg_revenue from lineorder, supplier where lo_suppkey = s_suppkey and s_nation = 'UNITED STATES' group by s_city order by avg_revenue desc

Q20: select c_city, s_city, count(*) from lineorder,customer, supplier where lo_custkey = c_custkey and lo_suppkey = s_suppkey and c_nation = 'UNITED STATES' and s_nation = 'UNITED STATES' group by c_city, s_city

Q21: select c_city, s_city, d_year, avg(lo_revenue) as avg_revenue from lineorder,customer, supplier, dates where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and c_nation = 'UNITED STATES' and s_nation = 'UNITED STATES' and d_year >= 1992 and d_year <= 1997 group by c_city, s_city, d_year order by d_year asc, avg_revenue desc

Q22: select c_city, s_city, d_year, count(*) from lineorder,customer, supplier, dates where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and c_nation = 'UNITED STATES' and s_nation = 'UNITED STATES' and d_year >= 1992 and d_year <= 1997 group by c_city, s_city, d_year order by d_year asc

Q23: select d_year, s_nation, p_category, sum(lo_revenue - lo_supplycost) as profit from lineorder,dates, customer, supplier, part where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_partkey = p_partkey and lo_orderdate = d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA' and (d_year = 1997 or d_year = 1998) and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2') group by d_year, s_nation, p_category order by d_year, s_nation, p_category

Q24: select d_year, s_nation, p_category, sum(lo_revenue - lo_supplycost) as profit from lineorder, dates, supplier, part where lo_suppkey = s_suppkey and lo_partkey = p_partkey and lo_orderdate = d_datekey and s_region = 'AMERICA' and (d_year = 1997 or d_year = 1998) and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2') group by d_year, s_nation, p_category order by d_year, s_nation, p_category

Q25: select d_year, s_nation, p_category, avg(lo_revenue - lo_supplycost) as avg_profit from lineorder, dates, customer, supplier, part where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_partkey = p_partkey and lo_orderdate = d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA' and (d_year = 1997 or d_year = 1998) and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2') group by d_year, s_nation, p_category order by d_year, s_nation, p_category

Q26: select d_year, s_nation, p_category, count(*) from lineorder, dates, customer, supplier, part where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_partkey = p_partkey and lo_orderdate = d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA' and (d_year = 1997 or d_year = 1998) and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2') group by d_year, s_nation, p_category order by d_year, s_nation, p_category

Q27: select d_year, s_nation, count(*) from lineorder,dates, supplier where lo_suppkey = s_suppkey and lo_orderdate = d_datekey and s_region = 'AMERICA' and (d_year = 1997 or d_year = 1998) group by d_year, s_nation order by d_year, s_nation

Q28: select s_nation, count(*) from lineorder,supplier where lo_suppkey = s_suppkey and s_region = 'AMERICA' group by s_nation order by s_nation

Q29: select d_year, s_nation, sum(lo_revenue) as revenue from lineorder,dates, supplier where lo_suppkey = s_suppkey and lo_orderdate = d_datekey and s_region = 'AMERICA' and (d_year = 1997 or d_year = 1998) group by d_year, s_nation order by d_year, s_nation

Q30: select sum(lo_revenue), p_brand from lineorder, part, supplier where lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand = 'MFGR#2221' and s_region = 'ASIA' group by p_brand order by p_brand

Main class of the MapReduce Input/Output Cost model

```
public class MainClass {
   public static void main(String[] args)
   {
   long Map;
   long Red;
   long tot;
   NewMap newMap = new NewMap();
   newMap.calculateTotalCost();
   Map = newMap.CMapRSJ ;
   NewRed newRed = new NewRed();
   newRed.calculateTotalCost();
   Red = newRed.CReduce ;
   tot = Map + Red ;
   }
}
```

References

Abouzeid A, Pawlikowski KB, Abadi DJ, Silberschatz A and Rasin A (2009). Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. In *Proceedings of the VLDB Endowment*, 2(1), pp. 922-933.

Agrawal D, Das S, and El Abbadi A (2011). Big Data and Cloud Computing: Current State and Future Opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology,* pp. 530-533, ACM.

Besse P, Garivier A, Loubes JM (2014). Big Data Analytics – Retour vers le Futur - 3 - De Statisticien à Data Scientist. In *Revue des sciences et technologies de l'information*, vol. 1633, pp. 1311.

Boukhalfa K (2009). De la conception physique aux outils d'administration et de tuning des entrepôts de données. In *Doctoral dissertation, ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechique-Poitiers*.

Boukorca A, Faget Z and Bellatreche L (2014). What-if Physical Design for Multiple Query Plan Generation. In *Database and Expert Systems Applications. Springer International Publishing,* pp. 492-506.

Brighen A (2012). Conception de bases de donn´ees volumineuses sur le cloud. In *Doctoral dissertation, Université Abderrahmane Mira de Béjaia*.

Brown PG (2000). Object-Relational Database Development: A Plumber's Guide. *Prentice Hall PTR, USA.*

Carstoiu D, Lepadatu E and Gaspar M (2010). Hbase-non SQL Database, Performances Evaluation. In *Int. J. Adv. Comp. Techn. (International Journal of Advanced Computer Technology)*, vol. 2, no 5, pp. 42-52.

Cattell R (2011). Scalable SQL and NoSQL Data Stores. In *ACM SIGMOD Record*, vol. 39, no 4, pp. 12-27.

Chaiken R, Jenkins B, Larson PA, Ramsey B, Shakib D, Weaver S and Zhou J (2008). Scope: Easy and efficient parallel processing of massive data sets. In *Proceedings of the VLDB Endowment,* vol. 1, no 2, pp. 1265-1276.

Cohen J, Dolan B, Dunlap M, Hellerstein JM, and Welton C (2009). MAD Skills: New Analysis Practices for Big Data. In *Proceedings of the VLDB Endowment*, vol. 2, no 2, pp. 1481-1492.

Cuzzocrea A, Bellatreche L and Song IY (2013). Data Warehousing and OLAP over Big Data: Current Challenges and Future Research Directions. In *Proceedings of the sixteenth international workshop on Data warehousing and OLAP*, ACM, pp. 67-70.

Cuzzocrea A, Song IY, Davis KC (2011). Analytics over Large-Scale Multidimensional Data: The Big Data Revolution!. In *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP*, ACM, pp. 101-104.

Dean J and Ghemawat S (2008). Mapreduce : Simplified data processing on large clusters. In *Communications of the ACM,* vol. 51, no 1, pp. 107-113.

DeWitt DJ, Gray J (1992). Parallel Database Systems: The Future of High Performance Database Processing. In *Communications of the ACM,* vol. 35, no 6, pp. 85-98.

Douglas K and Douglas S (2003). PostgreSQL: A comprehensive guide to building, programming, and administring PostreSQL databases. *Sams Publishing*, First Edition.

Eifrem E (2009). Neo4j—the benefits of graph databases. In *no : sql (east).*

Franco JM (1997). Le Data warehouse le Data mining. In *Informatiques magazine*, Ed Eyrolles.

Furtado P (2009). A Survey on Parallel and Distri buted Data Warehouses. In *International Journal of Data Warehousing and Mining (IJDWM),* vol. 5, no 2, pp. 57-77.

Gangarski S and Doucet A (2001). Entrepôts de données et Bases de Données Multidimensionnelles. *Paris: Hermès–Lavoisier*, Chapter 12 Bases de Données et Internet Modèles langages et systèmes, vol. 12, pp. 367-394.

Gardarin G (2003). Base de données. *Ed Eyrolles*, 5th edition.

George L (2011). HBase: The Definitive Guide. *O'Reilly Media, Inc.*, First Edition.

Gruska N and Martin P (2010). Integrating MapReduce and RDBMSs. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, IBM Corp., pp. 212-223.

Inmon WH (2005). Building the Data Warehouse. *John wiley & sons Publishing*, Fourth Edition.

Ketabchi MA, Mathur S, Risch T and Chen J (1990). Comparative Analysis of RDBMS and OODBMS: A Case Study. In *Proceedings of Compcon IEEE Computer Society International Conference*, San Francisco, CA.

Lammel R (2008). Google's MapReduce programming model — Revisited. In *Science of Computer Programming*, vol. 70, no 1, pp. 1-30.

Lee KH, Lee YJ, Choi H, Chung YD and Moon B (2012). Parallel Data Processing with MapReduce: A Survey. In *AcM sIGMoD Record,* vol. 40, no 4, pp. 11-20.

Manyika J, Chui M, Brown B, Bughin J, Dobbs R, Roxburgh C and Byers AH (2011). Big data: The next frontier for innovation, competition, and productivity. *McKinsey Global Institute*.

Matthew N and Stones R (2005). Beginning Databases with PostgreSQL. *Novice to Professional*, Second Edition.

McClean A, Conceição RC and O'Halloran M (2013). A Comparison of MapReduce and Parallel Database Management Systems. In *ICONS 2013, The Eighth International Conference on Systems*, pp. 64-68.

Mchome ML (2011). Comparison study between MapReduce (MR) and parallel data management systems (DBMs) in large scale data anlysis. In *Honors Projects Macalester College*, Paper 21.

Nance C, Losser T, Iype R and Harmon G (2013). NOSQL VS RDBMS - WHY THERE IS ROOM FOR BOTH. In *Proceedings of the Southern Association for Information Systems Conference*, Savannah, GA, USA, pp.111-116.

Narasimhan R and Bhuvaneshwari T (2014). Big Data – A Brief Study. In *International Journal of Scientific & Engineering Research*, Vol. 5, Issue 9.

Nykiel T, Potamias M, Mishra C, Kollios G and Koudas N (2010). Mrshare: sharing across multiple queries in mapreduce. In *Proceedings of the VLDB Endowment*, vol. 3, no 1-2, pp. 494-505.

Olston C, Reed B, Srivastava U, Kumar R and Tomkins A (2008). Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ACM, pp. 1099-1110.

Ordonez C (2013). Can we analyze big data inside a DBMS?. In *Proceedings of the sixteenth international workshop on Data warehousing and OLAP,* ACM, pp. 85-92.

Ordonez C, Song IY and Garcia-Alvarado C (2010). Relational versus non-relational database systems for data warehousing. In *Proceedings of the ACM 13th international workshop on Data warehousing and OLAP,* ACM, pp. 67-68.

Palla K (2009). A Comparative Analysis of Join Algorithms Using the Hadoop Map/Reduce Framework. In *Master of science thesis. School of informatics, University of Edinburgh.*

Pavlo A, Rasin A, Madden S, Stonebraker M, DeWitt D, Paulson E, Shrinivas L and Abadi DJ (2009). A comparison of approaches to large scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, ACM, pp. 165-178.

Ramakrishnan L, Mantha PK, Yao Y and Canon RS (2013). Evaluation of NoSQL and Array Databases for Scientific Applications. In *The International Conference for High Performance Computing, Networking, Storage and Analysis*.

Sabàu G (2007). Comparison of RDBMS, OODBMS and ORDBMS. In *Informatica Economică*.

Sabharwal N and Edward SG (2014). Big Data NoSQL Architecting MongoDB. In *CreateSpace Independent Publishing Platform*, USA, First Edition.

Sagiroglu S and Sinanc D (2013). Big Data: A Review. In *Collaboration Technologies and Systems (CTS), 2013 International Conference on. IEEE*, pp. 42-47.

Shvachko K, Kuang H, Radia S and Chansler R (2010). The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. IEEE,* pp. 1-10.

Strauch C, Sites ULS , Kriha W (2011). NoSQL Databases. In *Lecture Notes, Stuttgart Media University*.

Stonebraker M, Abadi D, DeWitt DJ, Madden S, Paulson E, Pavlo A and Rasin A (2010). Mapreduce and parallel dbmss : friends or foes ?. In *Communications of the ACM,* vol. 53, no 1, pp. 64-71.

Vouk MA. Cloud Computing – Issues, Research and Implementations (2008). In *Journal of Computing and Information Technology*, vol. 16, no 4, pp. 235-246.

Wang G and Chan CY (2013). Multi-Query Optimization in MapReduce Framework. In *Proceedings of the VLDB Endowment, 40th International Conference on Very Large Data Bases*, vol. 7, no 3.

White T (2009). Hadoop : The Definitive Guide. *O'Reilly Media Inc.*, First Edition.

Worsley JC and Drake JD (2002). Practical PostgreSQL. *O'Reilly and Associates Inc.*

Yui M and Kojima I. A Database-Hadoop Hybrid Approach to Scalable Machine Learning (2013). In *Big Data (BigData Congress), 2013 IEEE International Congress on. IEEE,* pp. 1-8.

Zaharia M, Konwinski A, Joseph AD, Katz R, Stoica I (2008). Improving MapReduce Performance in Heterogeneous Environments. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation OSDI*, Vol. 8. No. 4, pp.29-42.

Abstract

With the data volume which does not stop growing and the multitude of sources which led to of structures diversity, the classic tools of data management became unsuitable for processing. Hence the rapid development and change of the databases world, the evolution of data management solutions and the imposition of the Big date in our technological landscape which reflects both the data explosion and the recent capacity to handle it. This data management systems diversity presents a difficulty in choosing the best solution to interpret, protect and manage data according to the user's needs while preserving data availability.

In this work, we propose two contributions: The first is an implementation and a refinement of a cost model for MapReduce paradigm; Then, we propose an hybrid approach between two main categories of data management systems: classic DBMSs and NoSQL DBMSs. The idea is to integrate the ORDBMS PostgreSQL and MapReduce to perform OLAP queries in a goal of minimizing Input/Output costs in terms of the amount of data to manipulate, reading and writing throughout the execution process.

Keywords: MapReduce, RDBMS, NoSQL, integration, hybrid, cost, performance, OLAP.

Résumé

Avec le volume de données qui ne cesse de croître et la multitude de sources qui ont conduit à la diversité des structures, les outils classiques de gestion des données sont devenues impropres à satisfaire les nouveaux besoins. D'où le développement rapide et l'évolution du monde des bases de données, l'évolution des solutions de gestion des données et l'imposition de la Big Data dans notre paysage technologique qui reflète à la fois l'explosion des données et la capacité récente à les gérer. Cette diversité des systèmes de gestion de données présente une difficulté dans le choix de la meilleure solution d'interpréter, de protéger et de gérer les données en fonction des besoins de l'utilisateur, tout en préservant la disponibilité des données.

Dans ce travail, nous proposons deux contributions: La première est une implémentation et un raffinement d'un modèle de coût pour le paradigme MapReduce; Ensuite, nous proposons une approche hybride entre deux grandes catégories de systèmes de gestion de données: les SGBD classique et les SGBD NoSQL. L'idée est d'intégrer le SGBDR PostgreSQL et MapReduce pour exécuter des requêtes OLAP dans un but de minimisation les coûts d'entrée/sortie en terme de quantité de données à manipuler, lire et écrire tout au long du processus d'exécution.

Mots-clés: MapReduce, SGBDR, NoSQL, intégration, hybride, coût, performance, OLAP.