

ESCUELA TECNICA SUPERIOR DE INGENIEROS INFORMATICOS
UNIVERSIDAD POLITECNICA DE MADRID

DEPARTMENT OF DIGITAL SYSTEMS
UNIVERSITY OF PIRAEUS

Master of Science in

NETWORK-ORIENTED SYSTEMS

PARALLEL K-MEANS CLUSTERING ON TOP OF AN OLAP DATABASE

Master Thesis

By

Argyro Mavrogiorgou

Supervisors: Ricardo Jiménez Peris
Andriana Prentza

Madrid, 13/02/2015

The undersigned have examined the Thesis entitled "**Parallel K-Means Clustering on top of an OLAP Database**" presented by **Argyro Mavrogiorgou**, a candidate for the degree of **Master of Science in Network-Oriented Systems** and hereby certify that it is worthy of acceptance.

Date	Advisors name
Date	Committee member name
Date	Committee member name

Acknowledgements

I would like to express my gratitude to Ricardo Jiménez Peris, Marta Patiño Martínez and Andriana Prentza, for their decisive contribution into this Master Thesis. Also, I would like to thank the members of the committee Javier Soriano and María Pérez Hernández, for their time and their useful feedback.

More gratefully, I would like to thank my Professor and friend Dimos Kyriazis that gave me the chance to study for a few months abroad as an Erasmus student, giving me the opportunity to broaden not only my educational but also my cultural horizons.

Finally, I would like to thank my lovely family, my "twin" boyfriend and my new Italian, Argentinians and Spanish close friends, without whose love and support I could not have completed this long and difficult endeavor.

Dedicated to my mother, my father, my brother, and my boyfriend, the ones that are always there for me…

Με πολύ αγάπη, Λούλη

Abstract

Driven by advances in data collection and storage, increasingly large and high dimensional datasets are being stored. Without special tools, human analysts can no longer make sense of such massive volumes of data. As a consequence, intelligent Data Mining techniques are being developed either to semi-automate or totally automate the process of data mining. Clustering is an essential step in this process, as it reveals natural structures and identifies interesting patterns in the underlying data, by grouping high dimensional data together. In order to process this data, there have been proposed several clustering algorithms, such as the K-Means algorithm, which is well-known for its efficiency in clustering large datasets.

At the same time, while the Data Mining techniques are necessary for managing all these continuously growing data, the use of the distributed systems has become a major issue. Without them, the computing environments would not be able to manage such large-scales of data with an overwhelmingly high performance. Hence, several distributed frameworks have been created, aiming both to use the tolerance and the efficiency of the distributed systems and the priceless abilities of the Data Mining techniques.

This Thesis investigates the distributed framework of Apache Hadoop, a framework for large-scale data processing, where an entire ecosystem of self-standing projects has been built. Mahout is one of these projects, created to serve the various Data Mining issues. Thus, the Thesis studies about how the K-Means algorithm is used and implemented so as to process huge amounts of data using Mahout's machine learning libraries in combination with the Hadoop Distributed File System and the MapReduce key-value pair paradigm.

Additionally, in that Thesis it was developed a prototype project using a new parallel version of K-Means, so as to solve our main problem of performing parallel clustering on top of an OLAP distributed database. That algorithm uses Mahout's libraries to achieve scalability, an OLAP database for data storing purposes, and a new key-value pair concept for processing in parallel this data. This key-value pair concept was based on the MapReduce paradigm, but working in a different way. In the new concept, there are three different phases in the whole procedure (Map-Reduce-FinalReduce). Firstly, during the Map phase there are created numerous threads running in parallel different Map jobs, producing different key-value pairs. Then, in the Reduce phase each thread continues its corresponding Reduce job, processing its Map's output key-value pairs, without waiting for the other threads to finish their Map tasks. Finally, in the FinalReduce phase all the output key-value pairs of the Reduce phase of each thread are merged and the final key-value pairs are created.

Finally, summarized preliminary tests with experimental results of these implementations were stated, in order to verify that the developed prototype project is working properly, producing complete and reliable results.

Resumen

Debido a los avances que se están produciendo en la recolección y almacenamiento de información, grandes cantidades de datos están siendo almacenadas. Sin la ayuda de las herramientas adecuadas, los analistas no son capaces de procesar de manera correcta tal cantidad de datos. Como consecuencia, varias técnicas de procesamiento de datos inteligentes están siendo desarrolladas de tal manera que permite automatizar parcial o totalmente el proceso del estudio de los datos, data mining. Uno de los pasos fundamentales en el proceso de data mining es el clustering, ya que muestra estructuras y patrones que relacionan los datos a estudiar entre sí, permitiendo su agrupamiento. Para poder procesar estos datos se han propuesto diversos algoritmos de clustering, como por ejemplo el algoritmo k-medias, el cual es bien conocido por su eficiencia a la hora de agrupar grandes cantidades de datos.

Al mismo tiempo, para poder manejar de manera eficiente estos datos, que se encuentran en continuo crecimiento, se están utilizando sistemas distribuidos. Sin ellos, los entornos de procesamiento no sería capaces de manejar estas enormes cantidades de datos con un alto rendimiento. Esto, ha dado lugar a la creación de varios sistemas, con el objetivo de utilizar la tolerancia y la eficiencia de los sistemas distribuidos junto con las inestimables habiliades de las técnicas de data mining.

Esta tesis, se centra en la investigación de Apache Hadoop, una herramienta para el procesamiento de bases de datos altamente escalables, sobre la cual se han desarrollado un conjunto de proyectos. Mahout, es uno de esos proyectos, cuyo principal objetivo es ofrecer varias herramientas de data mining. De este modo, la tesis estudia como se puede utilizar e implementar el algoritmo de k-medias, mencionado anteriormente, en el procesamiento de grandes cantidades de datos utilizando las librerías de machine learning que ofrece Mahout junto con el HDFS y el paradigma de par clave-valor MapReduce.

Además, en esta tesis se ha desarrollado un prototipo utilizando una nueva versión paralela del algorítmo k-medias, resolviendo de esta manera nuestro principal problema de realizar el agrupamiento paralelo sobre una base de datos OLAP distribuida. Este algoritmo, utiliza las librería de Mahout para obtener la escalabilidad, una base de datos OLAP para el almacenamiento de los datos, y un nuevo concepto del par clave-valor para el procesamiento en paralelo de los datos. Este concepto del par clave-valor está basado en el paradigma MapReduce, pero funciona de manera diferente. En el nuevo concepto, hay tres fases diferentes dentro del proceso (Map-Reduce-FinalReduce). Durante la primera fase, fase Map, se crean numerosos procesos ejecutando en paralelo diferentes trabajos Map, produciendo diferentes pares clave-valor. A continuación en la fase Reduce, cada uno de los procesos continúa con su correspondiente trabajo Reduce, procesando los pares obtenidos en la fase anterior, sin esperar al resto de procesos a que terminen sus tareas Map. Finalmente, en la fase FinalReduce todas los pares clave-valor de cada una de las fases Reduce de cada proceso se mezclan de tal manera que forman los pares clave-valor definitivos. Finalmente, se realizaron una serie de experimentos cuyos resultados han permitido verificar que el prototipo desarrollado funciona adecuadamente y está produciendo resultados completos y fiables.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, first of all it is given an introduction paragraph about the subject that we are going to study in this Master Thesis, while it is described the main goals and the purpose of the Thesis, accompanied with the problem that we have to solve. Moreover, it is given a short explanation of the structure of the Thesis, in which are stated the containing chapters of it, with their corresponding contents.

1.1. Goals – Purpose of the Thesis

The evolution of information and telecommunication systems, has driven on the creation of a capable society, in providing new kinds and type of information. The gathered information is stored continuously, and as a result, a great amount of databases need to be created. The aforementioned fact is a modern phenomenon, which is observed as a need from the simplest to more complex issues of the people's daily lives.

So, it is an undeniable fact that we live in the data age. Web has been growing rapidly in size, as well as in scale, during the last 10 years and shows no signs of slowing down. Statistics show that every passing year more data gets generated than all the previous years combined.

Thus, the problem which arises is whether there is a manner of managing all these tons of data, which are continuously up-to-date, or not. Moreover, in combination with this, it seems to be very difficult to derive the necessary information from all this data. All these important issues, have driven the computer science into the creation of the Data Mining and the Machine Learning field, which include a series of techniques and methods based on various algorithms, in order to produce and classify useful insights and information for future or even immediate use.

As mentioned above, all the produced data is multiplied each year, with the disadvantage that the useful part of this data, gets lost and reduced, between the data's big volume. The field of Data Mining and the Knowledge Discovery not only from databases but also from other various data sources, expands rapidly in the computer science field, in order to come up against with all these problems.

More specifically, Data Mining has become exceedingly necessary because of the need for the creation of techniques and tools, which will aid in the analysis and interpretation of the quickly increasing data. At the same time, while the field of Data Mining tries to solve all these problems, another factor comes to goes with this, the use of the distributed systems, which are widely used in order to offer and serve large scale high performance computing environments, intended to manage all these huge amounts of data.

For that reason, in this Master Thesis, we are going to do an extensive study upon the field of Data Mining in combination with the concept of the distribution, by describing some very useful and commonly used data mining frameworks, methodologies, techniques and algorithms which are daily used, in order to extract and predict useful either structured or unstructured data insights.

More particularly, we are going to discuss about the Apache Hadoop MapReduce distributed framework, which is accompanied with the scalable machine learning framework of the Apache Mahout, a powerful systems' combination, for implementing data mining techniques into the 21st century's huge amounts of data. Thus, the *first challenge* that we have to face in this Master Thesis, is not only to deeply understand the usage and the significance of using all the aforementioned, but also to investigate and study the powerful combination of the two frameworks of the Apache Hadoop and Mahout, for classifying and sorting the today's data, with the same speed as its continuous development. More specifically, we study how Mahout implements its clustering techniques and algorithms (especially its K-Means clustering algorithm) for large-scale data processing, using as storage the Hadoop Distributed File System (HDFS) and the MapReduce key-value pair paradigm for processing this data in parallel.

The *second problem* that has raised and we have to solve in that Thesis, is whether there is a manner to perform parallel clustering techniques using the framework of Mahout for scalability purposes, an OLAP distributed database for storing (instead of the Hadoop Distributed File System (HDFS)), and a new key-value pair paradigm (instead of MapReduce). So, in order to solve this problem, it is developed a new parallel version of the K-Means clustering algorithm, which is making use all of the aforementioned for achieving scalability, distributed storing and parallel processing.

As for the storage part, the only difference is that we are going to use a different data storage repository, but concerning the part of the parallel processing of the data, things become more complicated. In particular, the difference between the original key-value pair concept of MapReduce and the new one, is that in the original concept of MapReduce all the Reduce jobs are beginning their execution after the successful

completion of all the running Map jobs. In the new concept, each created thread will run its own Map and Reduce jobs, and as a result of that, each Reduce job will have to wait only for the completion of its corresponding Map's job, and not for the completion of all the other Map jobs, that are performed by different threads. After this, when all the Reduce jobs of all the threads will have been successfully completed, a FinalReduce job will take place, and will merge the results of all the threads Reduce jobs.

As for the *general goals* from this thesis, these are listed below:

- Research and analysis of the basic meaning and usage of the Distributed Systems, in comparison with the Centralized Systems.
- Research of the Knowledge Discovery in Databases (KDD) area and its separate Data Mining field.
- Detailed study of the Clustering area, analyzing its significance and its several implementing techniques/algorithms.
- Analysis of the distributed Apache Hadoop framework, learning about its history and studying both for its individual core components (HDFS and MapReduce) and its whole created ecosystem (Mahout, Hive, Pig, etc.).
- Analysis of the Apache Mahout open source project which corporates with the Apache Hadoop project, in combination with its implementing algorithms, especially referring to its clustering algorithms.
- Analysis and understanding of the K-Means clustering algorithm, and implementation of it into two different projects. One for running the algorithm using Mahout's libraries on top of Hadoop Distributed File System (HDFS), and another one for running the algorithm using Mahout's machine learning libraries on top of an OLAP database.
- Evaluation and comparison of the results of the two implementations, in order to prove that the implementation of the K-Means Mahout algorithm on top of an OLAP database, is more efficient and productive than that of the HDFS.
- Quote of the conclusions which derive from the completion of the Thesis, accompanied with the future plans based on the research it was made.

1.2. Structure of the Thesis

This Master Thesis, consists of seven (7) different chapters, in which in the beginning of each chapter, there is always a small introduction, where it is summarized the content of it. More specifically, the chapters are separated as following:

- In **Chapter 1**, there is an introduction about the main subject of the Master Thesis and are explained the main targets and purposes that have been placed.
- In **Chapter 2**, takes place the background and the literature review of the corresponding theory that is required for the understanding and the illustration of the Thesis.
- In **Chapter 3**, the Apache Hadoop framework is described, in combination with its core components and its ecosystem.
- In **Chapter 4**, the Apache Mahout project is described, in combination with its data mining techniques, focusing especially on its clustering field.
- In **Chapter 5**, the K-Means clustering algorithm is described, followed by two different implementations, one for running the algorithm using Mahout's machine learning libraries on top of HDFS, and another one for running the algorithm using Mahout's libraries on top of an OLAP database.
- In **Chapter 6**, the comparison and the evaluation between the two implementations takes place.
- In **Chapter 7**, are described the conclusions and the future plans after the survey and the experiments that were made, as for the aforementioned subject.

Chapter 2

Background and Literature Review

In this chapter, is provided sufficient fundamental background information about the Distributed Systems and the Centralized Systems, and the widely spread area of Knowledge Discovery in Databases (KDD) with its field of Data Mining, with which we are going to deal with in this Thesis. More particularly, are given the definitions of the Distributed Systems and the Centralized Systems, accompanied with a brief comparison between them. In addition, the meaning of KDD and the Data Mining are explained, by providing the pertinent literature which is going to concern us in this Thesis, referring to the well-known field of Clustering.

2.1. Software Systems

Software Systems refer to all the separate files and programs that make up the computer's operating system. As for the files, these basically include libraries of functions, system services, drivers for printers and other hardware, system preferences, and other configuration files, whilst the programs include assemblers, compilers, file management tools, system utilities and debuggers. Since software systems run at the most basic level of the computers, they are called "low-level" Softwares. They generate the user interface and allow the operating system to interact with the hardware. So, in an abstract level, the software systems are the interfaces between the hardware and users' applications, and in a basic level they could be distinguished into two (2) different subcategories, the Centralized Systems and the Distributed Systems [1],[2],[3].

2.1.1. Centralized Systems

Centralized Systems are referring to systems where all or most of the parts of the processing/computing functions from different terminals/client machines, are performed on a central server. The central server, in turn, has to control all the separate machines directly, and as a result it is responsible for delivering application logic, processing and providing computing resources to these attached client machines. Thus, these machines run on a single common computer system and do not interact with other computer systems [4],[5],[6],[7].

The primary advantage of the centralized systems is their simplicity. More specifically, all the data from the different client machines is concentrated in one common place (the central host), where the centralized systems have the ability to easily manage all this data without any extra questions about its consistency or coherence. Moreover, centralized systems are relatively easy to secure, as they have only one central host that needs to be protected from general threats. Another advantage of centralized systems is that there is less backup complexity, as all the backups that have to be kept has to do only with the central host, and not with the separate client machines. Finally, as a result of all these, systems with a centralized model are lower capital and operational cost, as there is the minimal hardware at each site and less administrative overhead, as fewer resources needed since all equipment is in one location, this of the central host [8],[9].

However, centralized systems have some drawbacks, as well. The basic disadvantage of them is that if the central computer crashes, then the entire system will go down. To be more specific, all the processing/computing functions are done in only one place, as the central computer performs these functions and controls the remote terminals, which means that if the central computer goes down, everything does the same. Centralized systems are also often hard to extend, as the extra resources can only be added to the central system and not to the separate ones. Apart from this, centralized systems are often characterized as non-scalable, as their scalability is very subtle, as scale is clearly limited by the capacity of the server. Finally, in these systems there is no fault tolerance, and as a result the system can very easily shut down with a failure [6],[8],[9].

2.1.2. Distributed Systems

Distributed Systems are referring to information systems which consist of autonomous workstations that are connected through a network link, and appear finally to the users as a single, integrated computing facility. Each workstation has its own components, and it achieves to locate them and communicate with the other workstations by passing messages through the network. Thus, although each workstation is self-sustained for the most part, at the same time the data spread over multiple machines, and as a result, the different components of the workstations interact with each other, so as to achieve a common goal [8],[10],[11],[12],[13],[14],[15],[16].

Generally, distributed systems have a lot of advantages. The main benefit of them is that each workstation can work and 'survive' on its own, as it is independent from all the other workstations of the network. Moreover, in the distributed systems the components of the different workstations can easily be extended, as each workstation is an independent entity in the central network, thereby a better scalability will be provided among the workstations. Finally, distributed systems have the powerful benefit

of the very fast performance. More specifically, in the distributed systems there are many different workstations, each one of them with the ability of running a different instance of an application. As a result, at the same time run multiple instances from different workstations, increasing importantly the performance of the system [8],[10].

The advantages offered by distributed systems are also countered by some disadvantages. The main downside of these systems is the cost. To be more specific, not only they require additional hardware and software costs, but most certainly they require at least a partial onsite presence at each location, regardless of how many remote management components are in place. Moreover, distributed systems are not very safe, as the more hosts onsite they have, the greater the risk that the rest of the system will be threatened from the same 'enemies', unless special measures are taken in the case that one of the components fails. Another problem lies in the heterogeneity of their components, as the larger the geographical area over which a distributed system is used, the greater the probability that different types of technology will be incorporated. Finally, another consideration is the backup architecture, as the initial data backup processing does not be handled locally before being shipped or replicated [8],[10].

2.1.3. Centralized vs Distributed Systems

As mentioned above, both Centralized and Distributed Systems have their own characteristics, making them differ in a very significant way. More specifically, their basic characteristics are described in Table 1 and their architectures are shown in Figure 1.

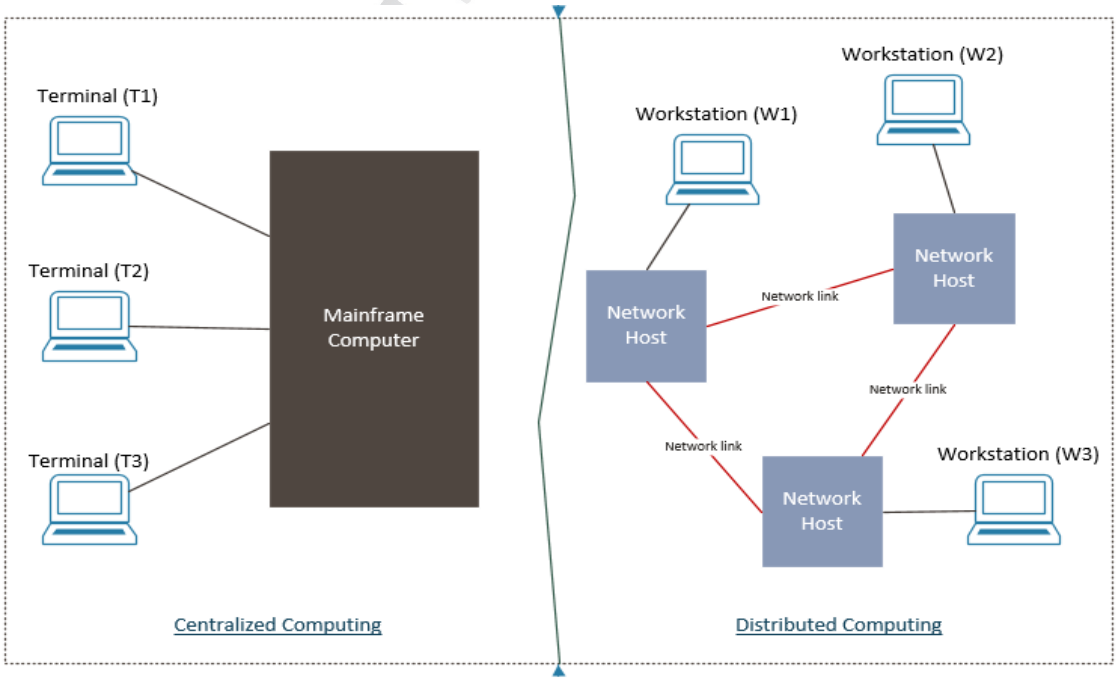


Figure 1 - Architecture of Centralized and Distributed Systems

Centralized Systems	Distributed Systems
• Contain one component with non-autonomous parts.	• Consist of multiple autonomous components.
• The resources are shared by users all the time.	• The resources of the components are not shared by all users.
• All the resources are accessible.	• Resources may not be accessible.
• Software runs in a single process, so several jobs are done on a particular central processing unit (CPU).	• Software runs in concurrent processes on different processors, so jobs are distributed among several processor.
• Are often built using homogeneous technology.	• Distributed systems may be built using heterogeneous technology.
• Have a single point of control and of failure.	• Distributed systems have multiple points of failure and control.

Table 1 - Centralized vs Distributed Systems

Taking into consideration all these characteristics of each type of systems, the advantages and the disadvantages between them are notable. More specifically, distributed systems offer a variety of advantages compared to centralized systems. First of all, distributed systems' applications can easily run simultaneously, having as a result to offer more benefits in terms of faster performance in comparison with the centralized solutions. Moreover, distributed systems can be extended through the addition of new components, thereby providing better scalability compared to centralized systems. Finally, in the distributed systems if one machine crashes, the system as a whole can still survive, in comparison with the centralized systems, where if one machine crashes, the whole system will crash as well.

However, these advantages of distributed systems are also countered by some disadvantages comparing to centralized systems. To begin with, the more components the distributed systems have, the greater risk it appears that the rest of the system will suffer. Moreover, the many components that make up a distributed system are potential sources of failures. Another problem with these systems lies in the heterogeneity of their components, as the larger the geographical area over which a distributed system is used, the greater the probability that different types of technology will be incorporated [10],[11],[14],[17],[18],[19].

2.2. Knowledge Discovery in Databases (KDD)

Driven by advances in data collection and storage, increasingly large and complex datasets are being stored in massive and high dimensional databases. Such enormous volumes of data clearly overwhelm traditional manual methods of data analysis. These methods can create informative reports from data, but cannot analyze the contents of these reports to focus on important knowledge [20],[21],[22]. Thus, the desire and need for information has led to the development of systems, tools, techniques and algorithms that can generate and collect all these massive amounts of data, creating meaningful information. These techniques and tools are the subject of the emerging field of Knowledge Discovery in Databases (KDD), an exploratory analysis for modeling large data repositories. The basic task of KDD is to extract information from low levels of data, so as to create a higher level of knowledge (abstract knowledge) with understandable patterns, or to discover a higher level of interpretation and abstraction different from those that were previously known. By simpler words, the main goal of KDD is to create more compact, more abstract and more useful information, by distinguishing from unprocessed data something that may not be obvious but is valuable or enlightening in its discovery [23],[24],[25],[26],[27],[28],[29],[30],[31].

Generally, modelling the investigated system and discovering relations from unprocessed data are the essence of KDD. Modern computer-based KDD systems self-learn from the previous history of the investigated system, formulating and testing hypotheses about the rules that the system obeys. When sententious and valuable knowledge about the system of interest has been discovered, it can be incorporated into some decision support system that will help the whole managing of the system. It should be mentioned that KDD is an interdisciplinary field, involving different fields of the computer science, such as database systems, statistics, machine learning, visualization, etc [22].

The KDD Process is both an iterative and an interactive process, consisting of several steps, involving numerous steps with many decisions being made by the user. Often the output of a step is feed back into the preceding step(s), and typically multiple iterations are required to extract high-quality knowledge. Thus, it is required to understand the process and the different needs and possibilities in each step. The interaction of the system with a domain expert through the monitoring of the loop is also necessary to ensure the usefulness and accuracy of the results. So, the overall process of KDD for finding and interpreting patterns from data, involves the repeated application of the following five (5) steps [20],[32]:

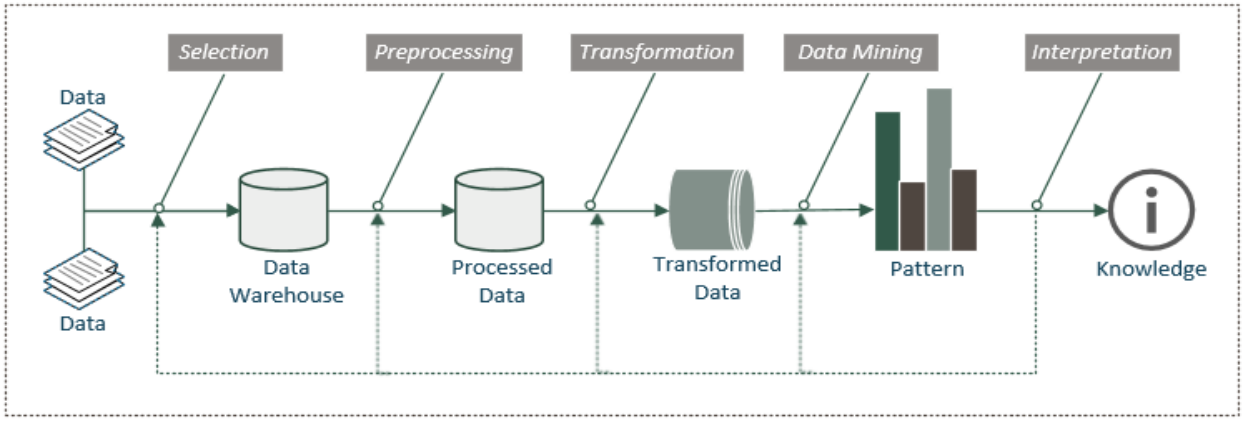


Figure 2 - KDD Process

1. *Selection*: The data to be mined may reside in different and heterogeneous data sources. Thus, the first step involves selecting the data relevant to the analysis task from various databases and integrating them into a coherent data store.

2. *Preprocessing*: During this stage the data is preprocessed, as it may have erroneous or missing values. In particular, the erroneous values are corrected or removed, while the missing ones are supplied or predicted in order to obtain only consistent data.

3. *Transformation*: The data are transformed into representations appropriate for mining tasks. Thus, during this stage data is organized, converted from one type to another and new or "derived" attributes are defined. This step can be crucial for the success of the entire KDD process, and it is usually very accurate and specific.

4. *Data Mining*: Data Mining is the core step of the KDD process, in which takes place the application of intelligent methods (clustering, classification, regression, etc.) with their corresponding algorithms, so as to extract the hidden information/knowledge from the transformed data. At this point the data is subjected to one or several data mining methods, as for example in order to develop an accurate classification model, it may firstly need to use clustering upon the data.

5. *Interpretation*: A data mining system has the potential to generate a large number of patterns but only a small fraction of them may be of interest. Thus, appropriate metrics are required to interpret and evaluate the interestingness of the results and identify those patterns that represent real knowledge and are of interest. So, actions at this stage could consist of returning to a previous step in the KDD process to further refine the acquired knowledge, or translating it into a form understandable to the user.

To sum up, the process "starts" with determining the KDD goals, and "ends" with the implementation of the discovered knowledge, where understanding and committing each step of the process is crucial for its success [23],[25],[26],[27],[31],[34],[35].

2.3. Data Mining

Having introduced the KDD process, we now focus on the data mining task, which as mentioned above, is the core component of the KDD process, and for that reason has received significant attention in the literature. The accessibility and abundance of data today makes data mining a matter of considerable importance and necessity. Given the recent growth of the field of data mining and its "big promises", it is not surprising that a wide variety of methods is now available.

More particularly, data mining comprises the core methods that enable someone to gain fundamental insights and knowledge from interesting and meaningful patterns from existing large amounts of data, which is stored either in databases or in other information repositories. So, data mining starts with getting data as an input, which can range from a simple array of a few numeric observations to a complex matrix of millions of observations with thousands of variables. In order to discover meaningful and useful structures from this data, it uses some specialized computational methods, such as clustering, classification, regression, etc. In other words, data mining involves the systematic analysis of large datasets using different automated methods. By probing data in this manner, it is possible to prove or disprove existing hypotheses or ideas regarding data or information, while discovering new or previously unknown information. In particular, unique or valuable relationships between and within the data can be identified in order to be used proactively for categorizing additional data. Through the use of exploratory graphics in combination with advanced statistics, machine learning tools and artificial intelligence, critical information can be mined from large repositories of data. In combination with this, and the fact that data mining processes can use data that has been gathered during many years and transformed into valuable knowledge, the field of data mining becomes increasingly powerful [23],[24],[31],[36],[37],[38],[39],[40].

In general, data mining can be applied to any kind of information repository as long as the contained data are meaningful for extracting knowledge. Thus, data mining is being put into use and studied for databases, including relational databases, data warehouses, transactional databases, unstructured and semi-structured repositories, time-series databases, etc. [42] So, it is an undeniable fact, that mining information and knowledge from all these different kinds of large databases has been recognized by many people as a key research topic in database systems and machine learning, and by many industrial companies as an important area with an opportunity of major revenues.

There are various data mining methods used for different purposes and goals in carrying out knowledge extractions from large databases, but it is useful to distinguish between them, two main types of data dining, the verification methods (the system verifies the user's hypothesis) and the discovery methods (the system finds new rules and patterns autonomously), which consist of different parts. More particularly, the individual parts of the verification and the discovery methods are being distinguished in Figure 3:

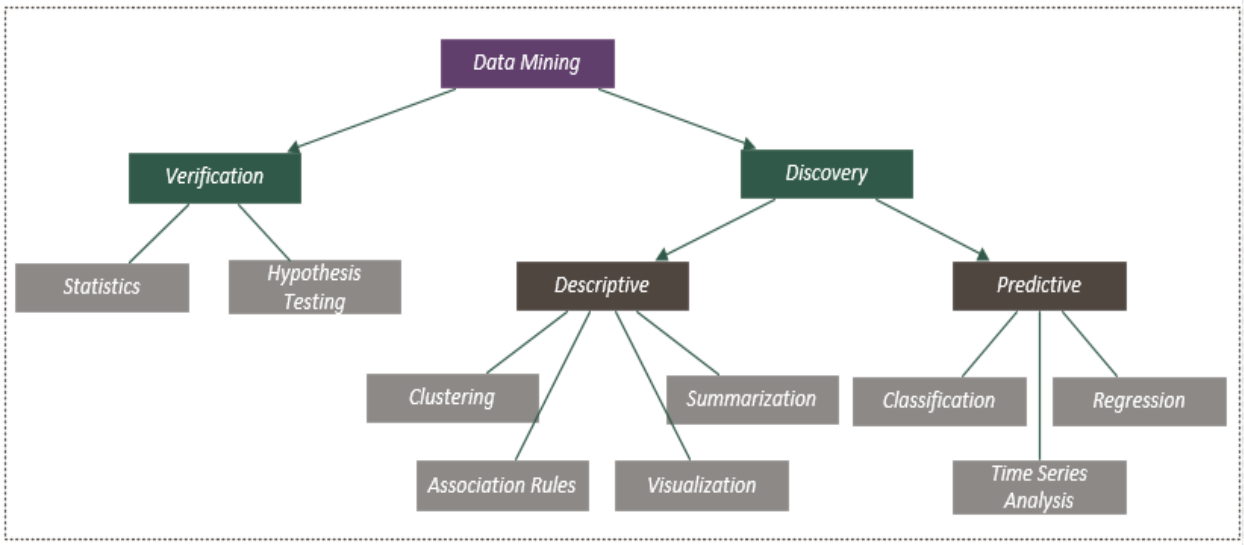


Figure 3 - Types of Data Mining Methods

In more detail, *verification methods* deal with the evaluation of a hypothesis proposed by an external source, like a user. These methods include the most common methods of traditional statistics and tests of hypotheses. These methods are less associated with data mining than these of the discovery methods, because most data mining problems are concerned with discovering a hypothesis, rather than testing an existing one [20],[22],23].

On the other hand, the *discovery methods* are those that automatically identify patterns in the large amount of data, and in a high level consist of the *descriptive* methods and the *predictive* methods.

At this points, it should be mentioned that although the boundaries between prediction and description are not sharp (some of the predictive models can be descriptive, to the degree that they are understandable, and vice versa), the distinction is useful for understanding the overall discovery goal.

As for the *predictive* methods, they are oriented to use some variables or fields from a database in order to predict unknown or future values of other variables of interest. To be more specific, this kind of methods aim to build a behavioral model, which obtains new and unseen samples and are able to predict values of one or more variables related to the sample. Moreover, they develop patterns which form the discovered knowledge in a way which is understandable and easy to operate upon. Classification, Regression and Time Series Analysis, are the most frequent types of tasks that are applied in predictive data mining.

- *Classification* is a well-known predictive supervised learning task, which aims to classify labeled data items into different classes, based on common properties (attributes) among a set of objects in a database. More particularly, each data instance consists of two different parts, a set of predictor attribute values and a goal attribute value. During classification, the dataset is divided into two mutually exclusive subsets, the training set and the testing set. In the training set the values of both the predictor attributes and goal attribute are available to the algorithm to learn a relationship between them, which is used subsequently to predict the class label of the data in the test set. The maximization of the classification accuracy rate in the test set is the main goal of learning [51],[52].

- *Regression* is a data mining task of aiming to learn patterns from examples and use the developed model to predict future values of the target variable. Whereas classification predicts categorical labels, regression models continuous-valued functions predicting numerical values. Usually a regression task begins with a dataset in which the target values are known, where the regression algorithm estimates the value of the target as a function of the predictors for each case in the build data. Afterwards, these relationships between predictors and target are summarized in a model, which can then be applied to a different dataset in which the target values are unknown [53],[60].

- *Time Series Analysis* is an ordered sequence of values of an attribute at equally spaced time intervals. There are three basic tasks performed in time series analysis. The first one is to obtain an understanding of the underlying forces and structure that produced the observed data, the second one is to find similarities or correlations between different time series and finally to fit a model and proceed to forecasting, monitoring or even feedback and feed-forward control.

As for the *descriptive* methods, they are oriented to data interpretation, which focuses on understanding the way of how the underlying data relates to its parts, and creating human-interpretable patterns describing this data. Clustering, Summarization, Association Rules and Visualization of data are the main applications of descriptive data mining for creating these patterns. The usefulness of this concept is that it give us the ability to generalize the dataset from multiple levels of abstraction, which facilitates the examination of the general behavior of the data, since it is impossible to deduce that from a large database.

- *Clustering* is a common descriptive unsupervised learning task, which aims to identify a finite set of categories to describe the data. It is the task of grouping a set of unlabeled objects in such a way that the objects in the same group called cluster, are more similar to each other than to those in the other clusters. More specifically, a similarity metric is defined between items of data, and then similar items are grouped together to form clusters. The grouping of data into clusters is based on the principle of maximizing the intraclass similarity and minimizing the interclass similarity. However the kind of grouping may be vary, as clustering can be achieved via various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them [43],[44],[45].

- *Summarization* involves methods for finding a compact description for a subset of data. Simple summarization methods such as tabulating the mean and standard deviations are often applied for data analysis, data visualization and automated report generation. Summarization can be viewed as a compression method of a given set of transactions into a smaller set of patterns, aiming to extract the maximum possible information [46],[47].

- *Association rules* refer to specific rules which can be viewed as a model that identifies specific types of data associations and are used in order to find frequent patterns, associations, correlations, or causal structures among these sets of data or objects in transaction databases, relational databases and other information repositories. These associations are often used in the retail sales management to identify items that are frequently purchased together, or to identify the best product to be offered with a current selection of purchased products [22],[48],[49],[50].

The research in this Master Thesis addresses the challenging descriptive unsupervised learning task of Clustering. When dealing with large and high dimensional datasets and databases, containing unknown and unlabeled data, clustering can be viewed as a type of data compression or summarization. Thus, the detailed data within the database is abstracted and compressed to smaller sets of class descriptions, where each one of these classes summarizes the characteristics of the data in each cluster. Although the idea of approximating a group of similar data points using its cluster descriptor loses fine details, it is very useful especially for large and high dimensional datasets, as it provides a simplification of the underlying data distribution, and it also helps to uncover hidden aspects of knowledge. So, in the terms of this Thesis, this was exact what we were looking for. A data mining method that would be able to manage effectively and efficiently huge amounts of data, with no previous knowledge and information about it. For that reason, in the next section of the Thesis, the meaning of Clustering and its individual components and tasks, are described in deep details.

2.4. Clustering

Clustering can be considered as the most important unsupervised learning data mining technique and one of the classical topics in the data mining field, in which the records in a dataset are organized into different logical groupings. These groupings are creating in such a way that records inside the same group are more similar than records outside the group. More specifically, in order to group these records, the data are partitioned into homogeneous clusters based on the values of their attributes. Thus, the records within a cluster have high similarity in some sense to each other, while records in separate clusters are more dissimilar in the same sense [20],[21],[22],[36],[43],[44],[45],[54].

Thereby, in order to be implemented clustering processes in the data, various clustering algorithms are used to organize the data, categorize it, compress it, etc. Often all these algorithms have a common approach so as to find the clusters' centers that will represent each cluster and their correspondingly assigned data objects, by defining a similarity metric between the data objects. Thus, the final expected result of the clustering will arise, as all the similar objects will be grouped together in order to form independent clusters [55].

However, clustering is not only used for grouping similar unlabeled data, but also used as a data compression and data preprocessing technique for supervised learning data mining tasks. Thus, it can be used as a standalone data mining tool to gain insight into the data distribution, or as a preprocessing step for other data mining algorithms operating on the detected clusters, such as classification [55].

So, it becomes clear, that clustering is an important element of exploratory data analysis. It is typically directed to study the internal structure of a complex dataset, which cannot be described only through the classical statistics. It is a very effective and efficient way of grouping data, as it helps us to construct meaningful partitioning of a large set of data based on a ”divide and conquer” methodology, which decomposes a large scale system into smaller components, in order to simplify design and implementation. As a data mining task, data clustering identifies clusters, or densely populated regions, according to a chosen distance measure, in a large, multidimensional dataset. Given a large set of multidimensional data points, the data space is usually not uniformly occupied by the data points. Data clustering identifies the sparse and the crowded places, and hence discovers the overall distribution patterns of the dataset. In order to decide what constitutes a good clustering, it is proved that there is no absolute “best” criterion which would be independent of the final result of the clustering. Consequently, it is the user which must choose this criterion in such a way that the result of the clustering will suit his needs. For example, people can be classified into a number of groups according to their monthly economic status or their annual alcohol consumption. These groupings will not necessarily capture the same individuals. The direction where the user is looking at the dataset depends, for example, on her/his background (position, education, culture etc.). So, it is clear that such things vary a lot among different individuals [43],[56],[57],[58].

Although the whole idea behind clustering is very simple, the successful completion of its tasks presume a large number of correct decisions and choices from several alternatives. More specifically, it has stated [59] that there appears to be at least nine (9) major elements in a cluster analysis study that someone has to keep in mind, before interpreting and evaluating the output of the final results:

- Data presentation.
- Choice of objects and variables.
- Normalization of variables.
- Choice of (dis)similarity measures.
- Choice of clustering criterion.
- Choice of missing data strategy.
- Algorithms and computer implementation.
- Number of clusters.
- Interpretation of results.

2.4.1. Challenges in Data Mining Clustering

Generally speaking, when someone decides to use clustering data mining methods for extracting useful information/knowledge from the data, he has to keep in mind that during the whole process, he will come up against with many challenges that he will have to resolve. In particular, clustering in high dimensional spaces is one of the biggest problems, due to the curse of dimensionality phenomenon, while the presence of irrelevant features and the obligation that each clustering algorithm has to satisfy several specific requirements, are some of the rest most common challenges [61],[62]:

As for the *curse of dimensionality*, it is basically used to refer to any problem in data analysis that results from a large number of variables. For clustering purposes, the most relevant aspects of the curse of dimensionality are the impact of increasing dimensionality on point proximity and density. In particular, distance-based clustering techniques depend critically on the measure of distance, and require that the objects within clusters are, in general, closer to each other than to objects in other clusters. Density-based clustering algorithms, in turn, require that the point density within the clusters must be significantly higher than the surrounding noise regions. Thus, under such circumstances, the data are "lost in space" and the effectiveness of clustering algorithms that depend critically on the measure of distance or density, waste rapidly with increasing dimensionality [63].

As for the *irrelevant features*, it is a common phenomenon that not all the dimensions of the data are relevant. Unfortunately, the presence of irrelevant features reduces any clustering tendency in the data. Thus, if all irrelevant features are pruned away, the points in each cluster come closer to each other, making easier the discrimination of clusters using either a distance or a density based criterion. However, feature selection techniques are susceptible to a substantial loss of information because different types of inter-attribute correlations may occur in different subsets of dimensions in different localities of the data. Therefore, it is vital for any kind of clustering algorithm to operate on the full dimensional space, in order not to be lost the data [62],[64].

Finally, an important challenge of clustering, is the fact that every clustering algorithm, has to obey in some specific requirements [22],[80]:

- *Handling high dimensionality*: Often, complex real-world concepts are accompanied by a large number of features. As a result of the sparsely filled space, the number of available points cannot grow exponentially with the dimensionality.

- *Irrelevant features - subspace clustering*: Often, especially in high dimensional spaces, not all dimensions are relevant, so it is vital for a clustering method to be able to detect clusters being embedded in subspaces possibly formed by different combinations of dimensions in different data localities.

- *Scalability*: The massive datasets, both in size and dimensionality, require highly scalable clustering algorithms, in order to be processed in the right way.

- *Clusters of arbitrary shape, size, density, and data coverage*: It is important to develop clustering algorithms that can detect clusters of arbitrary shape, size, density, and data coverage, gaining a deeper insight into the different correlations between the features.

- *Interpretability of the results*: Even the most advanced visualization techniques do not work well in high dimensional spaces, simply because the human eye-brain system is able to perform rough clustering only up to three dimensions. Therefore, it is quite essential to produce cluster that can be easily assimilated by an end-user.

- *Insensitivity to noise*: Most real-world databases contain noise and outliers that do not fit nicely into any of the clusters. The quality of the clustering results must not be affected by the presence of noise and outliers.

- *Insensitivity to initialization and order of input*: It is vital to develop clustering algorithms that produce similar quality results regardless of the initialization phase and the order in which input data are processed.

- *Minimal requirements for domain knowledge*: Clustering algorithms should have minimal requirements of auxiliary domain knowledge to determine the input parameters, since the former is rarely complete and consistent. Furthermore, the quality of the results must be relatively insensitive to the input settings.

- *Handling of different types of attributes*: Given the diversity of different kinds of data that are stored in the current databases (e.g. numerical, categorical and multimedia), real-world applications may require clustering data consisting of a mixture of data types.

2.4.2. Clustering Methods

There are many different ways to express and formulate a clustering problem, as a consequence, the obtained results and its interpretations strongly depend on the way the clustering problem was originally formulated. For example, the clusters or groups that are identified may be exclusive, so that every instance belongs in only one group, or may be overlapping, meaning that one instance may has been assigned into several clusters, or may be probabilistic, whereby an instance belongs to each group depending on a certain assigned probability, etc. Thus, it is very difficult to provide a categorization of clustering methods because these categories may overlap so that a method may have features from several categories. However, the clustering methods can be broadly classified into four (4) basic different types, as shown in Figure 4 [43],[54],[65],[66]:

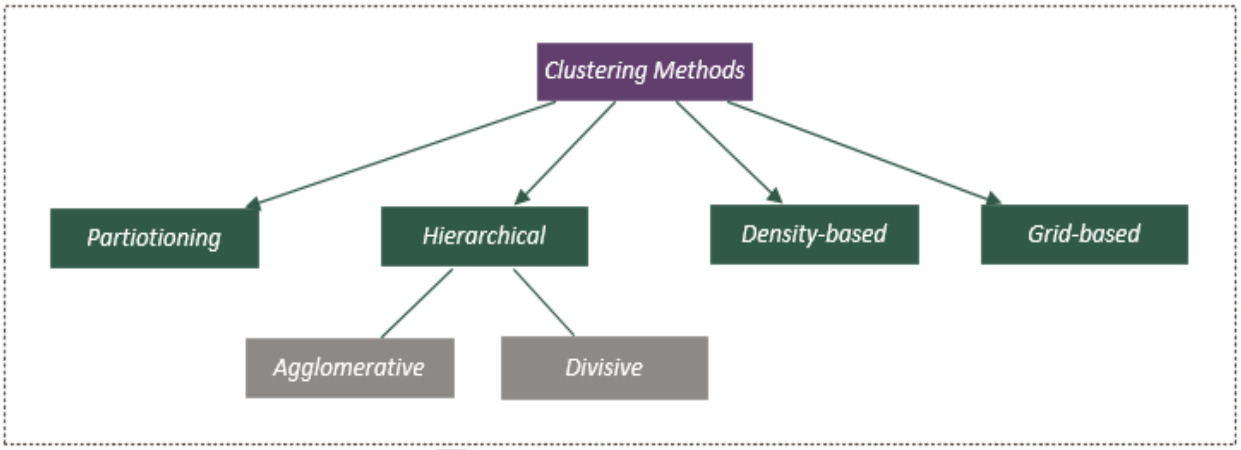


Figure 4 - Types of Clustering Methods

Partitioning methods: Given a set of n objects, a partitioning method has the ability to construct *k* partitions of the data, where each partition represents a cluster. In particular, it divides the data into k groups such that each group must contain at least one object. The basic partitioning methods typically adopt exclusive cluster separation, where each object must belong to exactly one group. The general criterion of a good partitioning is that objects in the same cluster are "close" or related to each other, whereas objects in different clusters are "far apart" or very different [44],[67],[68],[81].

Hierarchical methods: Hierarchical methods create a hierarchical decomposition of the given set of data objects, representing it by a tree structure, called dendrogram. More particularly, in this dendrogram every cluster node contains child clusters and sibling clusters, and each item is assigned to a cluster such that if we had *n* items then we would have *n* clusters. So, unlike partitioning methods create a single partition, hierarchical methods produce a nested sequence of clusters (the dendrogram).

Generally, hierarchical methods can be classified into two (2) subcategories, the agglomerative and the divisive, based on how the hierarchical decomposition is formed [43],[55],[65],[69],[70],[71]:

- The *agglomerative* approach, also called bottom-up approach, starts with each object forming a separate cluster, where clusters have sub-clusters, which in turn have sub-clusters, etc. More particularly, it starts by letting each object form its own cluster and iteratively merges cluster into larger and larger clusters, until all the objects are in a single cluster or certain termination condition is satisfied. The single cluster becomes the hierarchy's root. For the merging step, it finds the two clusters that are closest to each other, and combines the two to form one cluster [72].

- The *divisive* approach, also called top-down approach, works in a similar way to agglomerative clustering, but in the opposite direction. More specifically, starts with a single cluster containing all objects, and then successively splits resulting clusters until only clusters of individual objects remain [73].

Density-based methods: Density-based methods, have as a general idea to continue growing a given cluster as long as the density (number of objects or data points) in the "neighborhood" exceeds some threshold. Such methods can be used to filter out noise or outliers and discover clusters of arbitrary shape. Density-based methods can divide a set of objects into multiple exclusive clusters, or a hierarchy of clusters. Typically, these methods consider exclusive clusters only, and do not consider fuzzy clusters. Similar to hierarchical and partitioning methods, density-based techniques encounter difficulties in high dimensional spaces because of the inherent sparsity of the feature space, which in turn, reduces any clustering tendency [20],[54],[74],[75],[76],[80],[81].

Grid-based methods: Grid-based methods quantize the object space into a finite number of cells that form a grid structure, where all the clustering operations are performed on the grid structure. It is a fast processing time, being independent on the number of data objects and dependent only on the number of cells, which in turn, grows exponentially with the dimensionality. However, as the dimensionality increases more points map into individuals cells, thus making the use of summarized information decreasingly relevant to clustering. Finally, it should be mentioned that grid-based methods can be integrated with other clustering methods [77],[78],[79],[80],[81].

Table 2 describes in short the main characteristics of each clustering method:

TYPE OF METHOD	GENERAL CHARACTERISTICS
PARTITIONING	• Finds mutually exclusive clusters of spherical shape. • Distance-based. • May use mean to represent cluster center. • Effective for small-to medium size datasets.
HIERARCHICAL	• Clustering is a hierarchical decomposition. • Cannot correct erroneous merges or splits. • May incorporate other techniques like micro-clustering or consider object "linkages".
DENSITY-BASED	• Finds arbitrarily shaped clusters. • Clusters are dense regions of objects in space that are separated by low-density regions. • Each point must have a minimum number of points within its "neighborhood". • May filter out outliers.
GRID-BASED	• Uses a multi-resolution grid data structure. • Fast processing time.

Table 2 - General Characteristics of the Clustering Methods

2.4.3. Clustering Algorithms

There have been proposed hundreds of different data mining clustering algorithms, and are developed based on proximity between the records, density in the dataset, etc. However, not all of them provide models of their clusters and can thus not easily be categorized. Moreover, some of them integrate the ideas of several clustering methods, and as a result, sometimes it becomes difficult to classify a given algorithm as uniquely belonging to only one clustering method category. Regardless of the clustering method category that each algorithm belongs to, there is no objectively "correct" clustering algorithm, but the most appropriate clustering algorithm for a particular problem often needs to be chosen experimentally, unless there is a mathematical reason to prefer one cluster model over another. In Table 3, the most common algorithms of each clustering method are mentioned:

TYPE OF METHOD	ALGORITHMS
PARTITIONING	• *K-MEANS:* Creates k different clusters. • *EM:* Clusters are represented using a probability distribution. • *CLARA:* An implementation of K-MEANS in a subset of the dataset. • *CLARANS:* Combines the sampling techniques with K-MEANS.
HIERARCHICAL	• *CURE:* Clusters are represented by a fixed number of well-scattered points instead of a single centroid. • *CHAMELEON:* An implementation of CURE by using more elaborate merging criteria. • *BIRCH:* Compresses the data into many small sub-clusters. • *SLINK:* Provides a representation of the output clusters in a dendrogram.
DENSITY-BASED	• *DBSCAN:* Clusters are defined by a set of core objects with overlapping neighborhoods. • *OPTICS:* An extension of DBSCAN for local densities. • *DENCLUE:* Clusters are determined using density attractors.
GRID-BASED	• *WAVECLUSTER:* Maps the data into a user-specified multi-dimensional grid. • *STING:* Uses a hierarchical technique to divide the feature space into rectangular cells. • *CLIQUE:* Finds maximal cliques in graphs.

Table 3 - Clustering Methods' Developed Algorithms

In the terms of this Master Thesis, the partitioning K-Means algorithm was chosen to be used, which is perhaps the most popular clustering method in metric spaces. The main reason of choosing this algorithm, was the fact that in comparison with the existing hierarchical and other kind of algorithms, K-Means is a computationally faster algorithm, especially if the size of the data is extremely large. Thus, it can produce tighter clusters than the other algorithms, especially if the clusters are globular. The detailed function of K-Means and its characteristics will be explained in Chapter 5.

Chapter 3

Apache Hadoop

In this chapter, we provide general information about the Apache Hadoop distributed framework, in combination with its basic core components and its ecosystem. In particular, after describing the evolution of the Hadoop's framework and its usage, we analyze its core components (HDFS and MapReduce). After that, we state the most commonly used platforms which cooperate with Hadoop, which have created the general meaning of the Hadoop ecosystem.

3.1. History of Hadoop

Named after a toy elephant belonging to developer Doug Cutting's son, over the past decade Hadoop has proven to be the little platform that could. Hadoop started out as a subproject of Nutch, a complete web search engine developed by Doug Cutting, as well.

Around 2004, Google released two academic papers describing Google technology, the Google File System (GFS) and MapReduce, two technologies that provided together a platform for processing data on a very large scale in a highly efficient manner.

More specifically, at the 2003 Symposium on Operating Systems Principles (SOSP) and in the companion article, The Google File System, S. Ghemawat, H. Gobioff, and S. Leung described GFS as follows: *"...a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.*" [83]

Slightly more than a year later, in 2004, S. Ghemawat and J. Dean unveiled MapReduce at the Symposium of Operating Systems Design & Implementation (OSDI). In MapReduce: Simplied Data Processing on Large Clusters, they presented MapReduce as follows: *"...a programming model and an associated implementation for processing and generating large datasets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.*" [83]

Doug Cutting immediately saw the applicability of these technologies to Nutch, and his team implemented the new framework, porting Nutch to it, and boosting as a results its scalability. It started to handle several hundred million web pages and could run on clusters of dozens of nodes. Doug realized that a dedicated project to flesh out the two technologies was needed to get to web scale, and Hadoop was born. Therefore, Hadoop is an open source platform that provides implementations of both the MapReduce and GFS technologies and allows the processing of very large datasets across clusters of low-cost commodity hardware [82],[84].

Yahoo! hired Doug Cutting in 2006 and quickly became one of the most prominent supporters of the Hadoop project. The platform became crucial to the operations of the company's data science team, while in the meanwhile, Hadoop was embraced within the open source community and by developers at companies such as Google and Facebook, accelerating its update cycle and lending the platform additional credibility and battle-tested stability. Two years later, Hadoop achieved the status of an Apache Top Level Project, and since then, the platform has flourished a lot, adopting by considerable development resources and many contributors, both academic and commercial, such as Facebook, Ebay, Twitter, LinkedIn, and New York Times [82],[84],[85],[86],[87],[88],[89].

In general, Hadoop's main goal is to achieve massive parallel processing by taking advantage of big data, by using lots of inexpensive commodity servers [90],[91],[92]. Thus, Hadoop is not a simple platform, but an open source framework for writing and running distributed applications that process large amounts of data, offering to these applications the following very significant benefits [90]:

- *Accessibility:* Hadoop runs on large clusters of commodity machines or on cloud computing services.

- *Simplicity:* Hadoop allows users to quickly write efficient parallel code.

- *Robustness:* Hadoop is architected with the assumption of frequent hardware malfunctions as it runs on commodity hardware, having the ability to handle most such failures.

- *Scalability:* Hadoop scales linearly to handle larger data by adding more nodes to the cluster.

In more details, Hadoop's accessibility and simplicity give it an edge over writing and running large distributed programs. Even college students can quickly and cheaply create their own Hadoop cluster. On the other hand, its robustness and scalability make it suitable for even the most demanding employees and employers. Thus, these features made Hadoop popular in both academic and commercial contributors.

Figure 5 illustrates how users interact with a Hadoop cluster, where a Hadoop cluster is a set of commodity machines networked together in one location. Data storage and processing all occur within this "cloud" of machines, where different users can submit computing "jobs" to Hadoop from individual clients, obtaining their results [82].

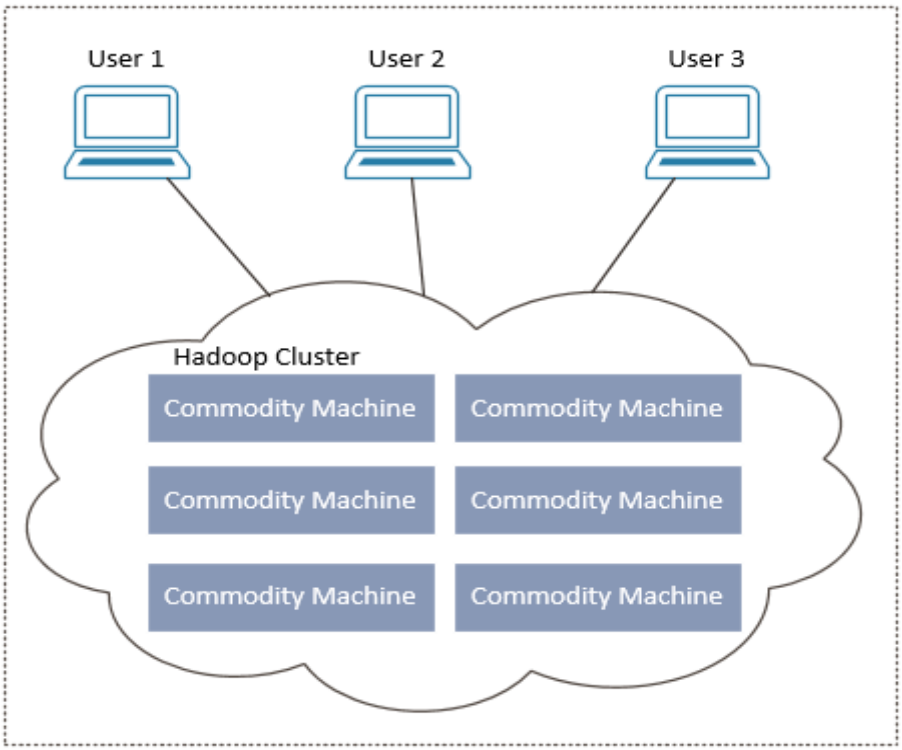


Figure 5 - Users' interaction with Hadoop Clusters

3.2. Hadoop Architecture

The Hadoop framework, consists of a distributed master-slave architecture, which contains two (2) different main components: a reliable distributed file system called Hadoop Distributed File System (HDFS) and a high-performance parallel data processing engine called Hadoop MapReduce, which keep their roots from Google's File System (GFS) and MapReduce, respectively. The most important aspect of Hadoop, is that both HDFS and MapReduce are designed having each other in mind, and each are co-deployed such that there is a single cluster, providing in this way the ability to move computation to the data, but not the other way around. Thus, the storage system is not physically separate from the processing system. So, the powerful combination of HDFS and MapReduce provides a software framework for processing huge amounts of data in parallel, on large clusters of commodity hardware in a reliable, fault-tolerant manner [83],[84],[93],[94].

As for its first component, *HDFS*, it provides scalable, reliable and low cost storage, by storing files across a collection of servers in a cluster and creating multiple replicas of data blocks onto multiple nodes, in order to safeguard the environment from any potential data loss. HDFS ensures data availability by continually monitoring the servers in a cluster and the blocks that they manage [95],[96],[97],[98].

As for the second component, *MapReduce*, it is a programming model and software framework for writing applications that rapidly process vast amounts of data in parallel. Using this model, Hadoop enables the users to explore and analyze complex datasets by utilizing customized analysis functions/scripts [95],[96],[97].

It should be mentioned that MapReduce has undergone a complete overhaul in hadoop-0.23 and from then there is the MapReduce 2.0 or YARN. Apache Hadoop YARN is a sub-project of Hadoop at the Apache Software Foundation introduced in Hadoop 2.0 that separates the resource management and processing components and provides a more general processing platform that is not constrained to MapReduce [91].

So, the way that those two (2) basic components cooperate in the terms of the Hadoop architecture, is that Hadoop using HDFS splits the files with the data into large blocks (default 64MB) and distributes these blocks amongst the nodes in the cluster. To process the data, MapReduce transfers code (specifically Jar files) to nodes that have the required data, and process it in parallel. In the following sections, we will deeply study about those two components and their functions [92].

3.2.1. Hadoop Distributed File System (HDFS)

As mentioned above, Hadoop Distributed File System (HDFS) is a distributed file system inspired by GFS from Google, which runs on low cost commodity hardware in a fault tolerant manner to redundantly store terabytes and larger amounts of data. Therefore, it offers a way to store large files across multiple machines, rather than requiring a single machine to have disk capacity equal to or greater than the total size of the files.

As shown in Figure 6, HDFS has a master-slave architecture, consisting of [83],[93]:

- *NameNode:* It is the master node of the system. It maintains the name system (directories and files) and manages the blocks which are present on the DataNodes.
- *DataNodes:* They are the slaves of the system, which are deployed on each machine and provide the actual storage. They are responsible for serving read and write requests for the clients.
- *Secondary NameNode:* It is responsible for performing periodic checkpoints. In the event of NameNode failure, it is able to restart the NameNode using the checkpoint.

It is worth mentioning that HDFS is built using the Java programming language, thus any machine that supports Java can run the NameNode or the DataNode software.

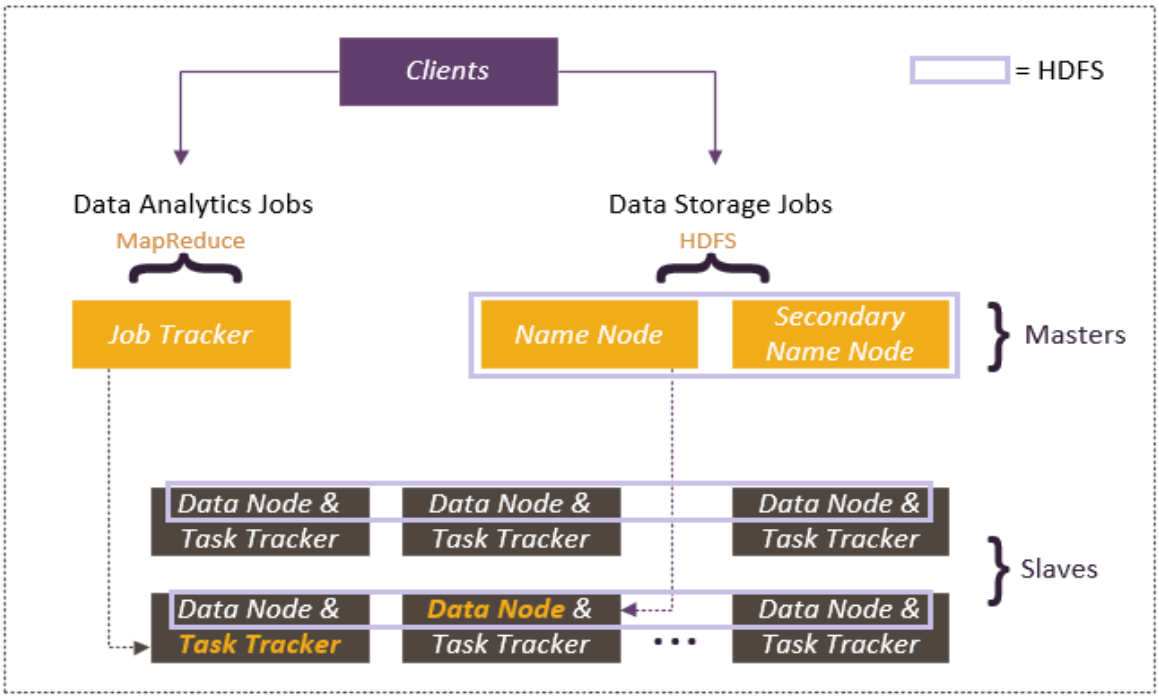


Figure 6 - HDFS master-slave Architecture

Generally, the way that HDFS works is quite simple. In particular, it divides the data into fixed-size (64MB or larger) blocks and spreads them across all DataNodes, which (usually) exist in each cluster. Then, the NameNode maintains all the metadata associated with the data stored on the DataNodes, being responsible for coordinating access and replication of the data, and executing namespace operations like opening, closing, and renaming files and directories. The NameNode stores all of this information in RAM for fast access, but keeps an image on disk in case of failure. If the NameNode fails, this image may be used to restart the NameNode, creating a secondary NameNode. The DataNodes themselves store no information about the data they contain, but they simply store each block of data as a separate file on the local file system and are responsible for performing block creation, deletion, and replication upon instruction from the NameNode. When the NameNode splits and distributes data, it does not respect record boundaries in the data. It is often the case that the first and last record in a split are truncated. In the case of truncated records DataNodes will transfer the remaining portion of the truncated records to the appropriate machines during runtime [99],[100],[101].

So, in short, HDFS requires a NameNode process to run on one node in the cluster and a DataNode service to run on each slave node that will be processing data. When data is loaded into HDFS, the data is replicated and split into blocks that are distributed across the DataNodes. The NameNode is responsible for storage and management of the metadata, so that when MapReduce or another execution framework requests for the data, the NameNode informs it where the needed data are located [83].

The HDFS architecture is designed to be a write-once, read-many system. Once a file is closed, it is replicated across the cluster, and cannot be written again. The system is optimized for high sustained throughput streaming reads. This means that it is faster to read an entire 64MB block and filtering out information that is not needed, than it is to do a low latency seek to specific locations in that block to read the same information.

It should be mentioned, that the data in the HDFS can be accessed using the command line, a Java API, or a C language wrapper for that same API. As for the Java API, it is very similar to the API's used for normal file access, allowing for all of the familiar operators, including implementations of input and output streams, which allow buffered reads and writes using the standard Java API. Finally, from the users' perspective, the data access is similar to that of a normal file system, but the details of the data storage in the HDFS discussed in this section are hidden from the user [91],[100].

3.2.2. MapReduce Framework

As mentioned above, MapReduce is a framework for performing distributed data processing, inspired by Google's MapReduce, constructed for running on top of HDFS. The main idea of the framework relies on the map and reduce functions commonly used in functional programming. For that reason, in MapReduce, each job has a user-defined "Map" phase which is a parallel, share-nothing processing of input, followed by a user-defined "Reduce" phase where the output of the map phase is aggregated. Typically, HDFS is the storage system for both input and output of the MapReduce jobs [93],[99].

As shown in Figure 7, MapReduce has a master-slave architecture, like HDFS, but consists of two other types of nodes that control its job execution process [91],[93],[99],[102]:

- *Job Tracker:* It is the master of the system and coordinates all the jobs that have to run on the system by scheduling tasks to run on Task Trackers.
- *Task Trackers:* They are the slaves of the system, which are deployed on each machine, and are responsible for running the assigned map and reduce tasks from the Job Tracker. It should be mentioned, that MapReduce Task Trackers run on the same set of nodes that HDFS DataNodes run on.

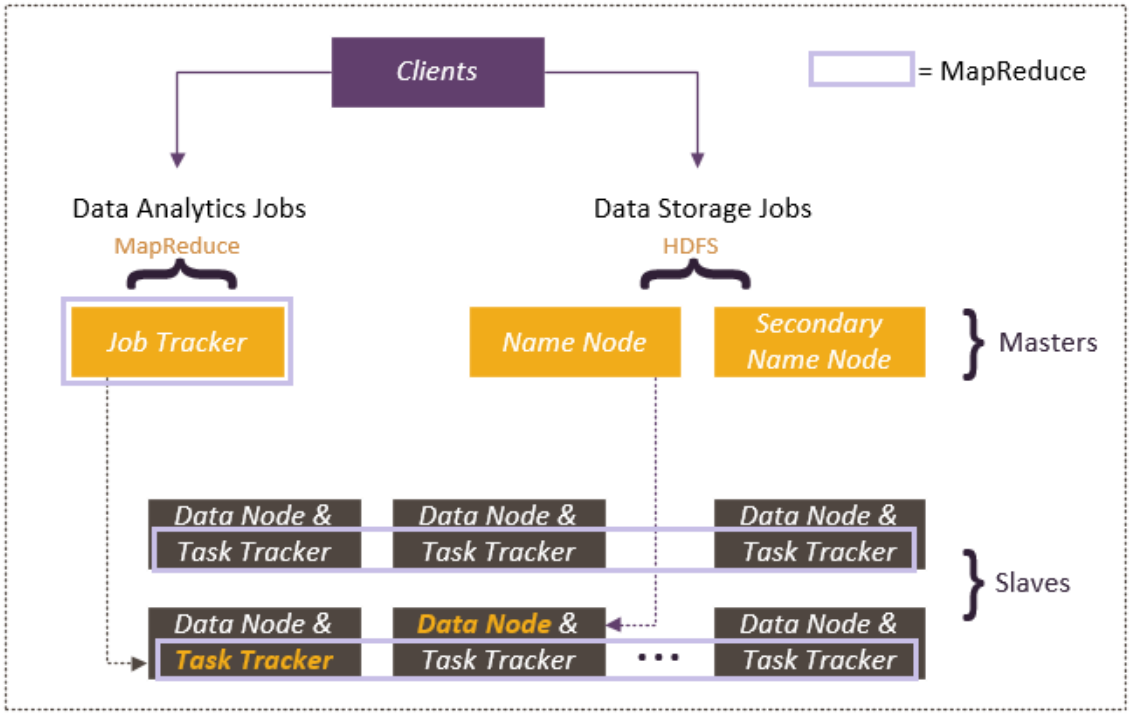


Figure 7 - MapReduce master-slave Architecture

Figure 8 illustrates the overall flow of a MapReduce process, in combination with the functions of the Job Tracker and the Task Trackers:

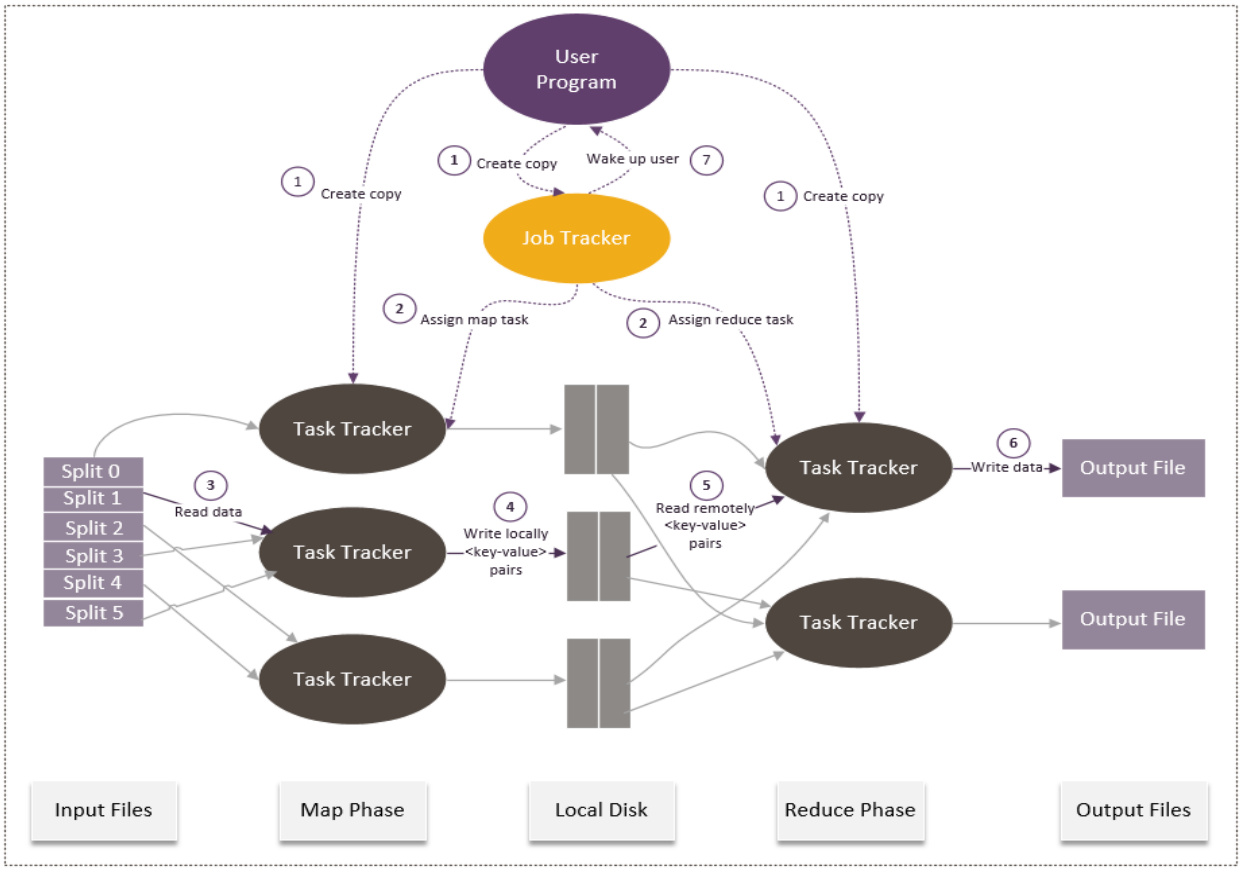


Figure 8 - MapReduce Process

(1) First of all, the procedure begins when a user program calls the MapReduce function, which stores the dataset in the HDFS by splitting it into blocks of 64MB, and then starts up many copies of the program on a cluster of machines.

(2) One of these copies is the Job Tracker and the rest of these are the Task Trackers that are assigned work by the Job Tracker. Thus, the Job Tracker creates "M" map tasks and "R" reduce tasks, and assigns them to the existing Task Trackers, by picking idle Task Trackers and assigning to each one of them a different map or reduce task.

(3) Then, every Task Tracker to whom was assigned a map task reads the contents of the corresponding block and creates for its data key-value pairs. The intermediate key-value pairs produced by the Map function are buffered in memory.

(4) Periodically, the buffered pairs are written to the local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the Job Tracker, who is responsible for forwarding these locations to the reduce Task Trackers.

5 When a reduce Task Tracker is notified by the Job Tracker about these locations, it uses remote procedure calls to read the buffered data from the local disk. So at that time, the reduce Task Trackers have to wait until the completion of the map Task Trackers. Thus, when a reduce Task Tracker has read all the intermediate data, it sorts it according to their keys, so that all occurrences of the same key are grouped together. If the amount of intermediate data is too large to fit in memory, an external sort is used.

6 The reduce Task Tracker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce function is appended to a final output file for this reduce partition.

7 When all the map and reduce tasks have been completed, the Job Tracker wakes up the user program and the output of the MapReduce execution is available in the final output file.

So in simple words, the MapReduce paradigm comprises two (2) basic sequential phases: The map phase and the reduce phase. In the map phase, the input is a set of key-value pairs, where a user-defined map function is executed over each key-value pair, in order to generate a set of intermediate key-value pairs. Then, in the reduce phase, these intermediate key-value pairs are grouped by key and the values are combined together according to the user-defined reduce function. It is also possible that no reduce phase is required, given the type of operation coded by the user [102].

It should be mentioned, that in order to detect a Job Tracker the failures, it pings every Task Tracker periodically. If no response is received from a Task Tracker in a certain amount of time, then the Job Tracker marks the Task Tracker as failed. Any map tasks completed by a Task Tracker are reset back to their initial idle state, and therefore become eligible for scheduling from other Task Trackers. Similarly, if any map task or reduce task in progress fail, then the corresponding Task Tracker also resets to idle state and becomes eligible for rescheduling. When a failure occurs, the completed map tasks are re-executed, because their output is stored on the local disk of the failed machine and is therefore inaccessible, whilst the completed reduce tasks do not need to be re-executed, since their output is stored in a global file system [85],[98],[103],[104],[105].

Finally, it is worth mentioning that the minimum job execution time is estimated to be around thirty seconds to a minute, although the typical use case for Hadoop jobs take hours or days to complete on thousands of machines. This execution time multiplies when the algorithm requires multiple Map/Reduce iterations, and is an important factor to consider when choosing whether to use the Hadoop framework or not for a given job.

3.3. Hadoop Ecosystem

When Hadoop was firstly released by Apache, consisted mainly of HDFS and MapReduce, but soon it became clear that Hadoop was not simply a single application. It was a platform around which an entire ecosystem of capabilities could be built. Since then, hundreds of self-standing software projects have sprung into being around Hadoop, each addressing a variety of problems and meeting different needs. Many of these projects were begun by the same people or companies who were the major developers and early users of Hadoop, whilst others were initiated by commercial Hadoop distributors. The majority of these projects now share a home with Hadoop at the Apache Software Foundation, which supports open source software development and encourages the development of the communities surrounding these projects.

As someone would expect, these projects are not meant to be used all together as parts of a single organism, but some may even be aiming to solve the same problems in different ways. As a result, the functionalities of these projects may overlap with other projects in the Apache foundation project. Hence, the choice of the components that have to be integrated in order to form an overarching system is left to the discretion of its users. However, it should be mentioned that all these projects have a basic common part, meaning that they seek to tap into the scalability and power of Hadoop, either by building on top of Hadoop or working very closely with it [92],[102],[105],[106],[107].

As a result, all these projects have given rise to an ecosystem, so-called as "Hadoop Ecosystem" that focuses on large-scale data processing, and often users can use several of these projects in combination to handle their use cases. Following, are introducing several key projects in the Hadoop ecosystem, as illustrated in Figure 9:

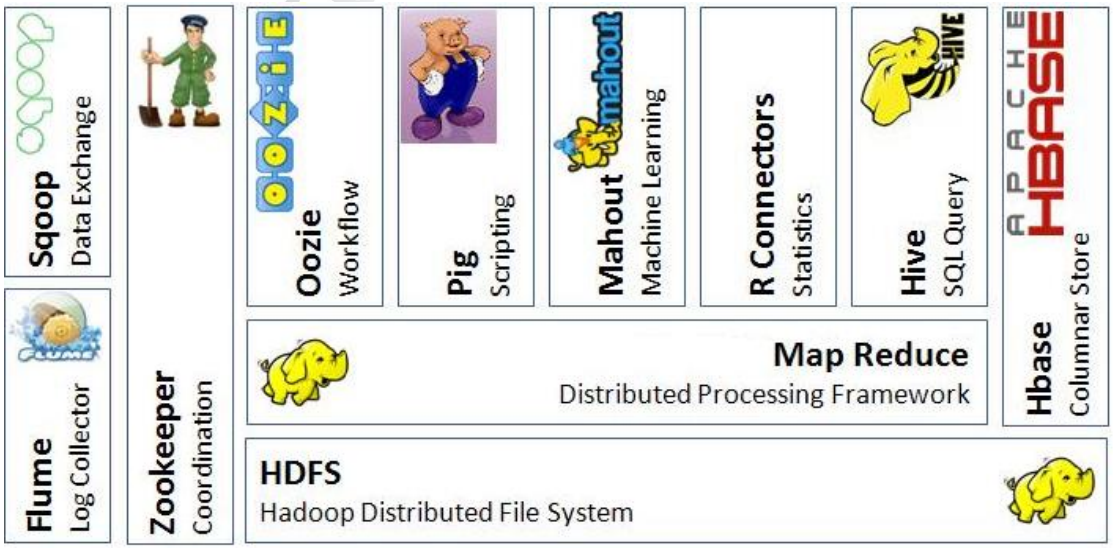


Figure 9 - Hadoop Ecosystem

Pig is a platform with a high-level query language built to handle large datasets. In particular, Apache Pig uses the data flow language Pig Latin, in order to support relational operations such as join, group and aggregate and scale across multiple servers simultaneously. Moreover, Pig consists of a compiler that produces sequences of MapReduce programs so as to be executed within Hadoop. Analytics on sample data, running complex tasks that collates multiple data sources are some of the use cases that can be handled using Pig [89],[102],[107],[108],[109],[110].

Hive is a data warehouse infrastructure built on top of Hadoop that connects the gap between SQL based RDBMS and NoSQL based Hadoop. In particular, Hive was created to make it possible for analysts with strong SQL skills but not so strong Java programming skills, to run queries on the huge volumes of data, in order to extract patterns and meaningful information. It provides an SQL-like language, called HiveQL, while it maintains full support for MapReduce, as each Hive query is converted to MapReduce tasks. Thus, Hive reduces the difficulty of a programmer to write MapReduce jobs if he/she knows SQL [102],[107],[108],[111],[112],[113].

ZooKeeper is a distributed open-source coordination service. In particular, it provides a simple set of primitives that distributed applications can build upon to implement higher level services for synchronization, configuration maintenance, grouping and more. ZooKeeper steps up performance and availability of a distributed system such as a Hadoop cluster and protects the system from coordination errors such as deadlocks. In general, the motivation behind ZooKeeper is to relieve the distributed applications from the responsibility of implementing coordination services from scratch [102],[107],[108],[110],[113].

Mahout is a suite of scalable machine learning libraries implemented on top of Hadoop, used mainly for predictive analytics and other advanced analysis. There are currently four (4) main groups of algorithms in Mahout: recommendations, classification, clustering and frequent itemset mining, which belong to the subset that can be executed in a distributed fashion, as they have been written to be executable in MapReduce. Although at the beginning, it was primarily set out to collect algorithms for implementation on MapReduce, the last months it has begun implementing algorithms on other platforms which are more efficient for data mining, such as Spark [107],[108],[110],[111],[113].

HBase is a NoSQL distributed column-oriented database that runs on top of HDFS, performing random read/write operations. Because of its column-based architecture, HBase enables high-speed execution of operations performed over similar values across massive datasets. It can also be combined with MapReduce to ease bulk operations such as indexing or analysis, but it depends on ZooKeeper and by default it manages a ZooKeeper instance as the authority on cluster state. HBase hosts vital information such as the location of the root catalog table and the address of the current cluster Master [102],[107],[108],[110],[112],[113].

HCatalog is a metadata and table storage management service for HDFS. HCatalog depends on the Hive metastore and exposes it to other services such as MapReduce and Pig, with plans to expand to HBase using a common data model. HCatalog's goal is to simplify the user's interaction with the data that are stored in the HDFS and enable data sharing between tools and execution platforms [102],[110].

Avro is a framework for performing remote procedure calls and data serialization. In the context of Hadoop, it can be used to pass data from one program or language to another (e.g. from C to Pig). It is particularly suited for use with scripting languages such as Pig, because data is always stored with its schema in Avro, and therefore the data is self-describing [102],[108].

Flume is a tool that aggregates streaming data from different sources and adds them to a centralized datastore for Hadoop cluster, such as HDFS. In particular, it harvests, aggregates and moves large amounts of log data in and out of Hadoop. In essence, Flume is what takes the data from the source and delivers it to Hadoop. Possible sources for Flume include Avro, files, and system logs, and possible sinks include HDFS and HBase [102],[107],[108],[111].

Sqoop is the latest Hadoop framework that provides an efficiently bi-directional data transfer between HDFS and structured data repositories, such as relational databases, enterprise data warehouses and NoSQL systems. For example, Sqoop enables two-way import/export of bulk data between HDFS/Hive/HBase and relational or structured databases. Unlike Flume, Sqoop helps in data transfer of structured datasets [102],[107],[108],[110],[113].

Chukwa is a data collection and analysis system built on top of HDFS and MapReduce. Tailored for collecting logs and other data from distributed monitoring systems, Chukwa provides a workflow that allows for incremental data collection, processing and storage in Hadoop. It is included in the Apache Hadoop distribution but as an independent module [102],[108].

Chapter 4

Apache Mahout

In this chapter, we provide general information about the Apache Mahout project and its components. In particular, at first place the Mahout software and its history are described, and then the machine learning techniques/methods which are implemented in Mahout (Recommender Engines, Classification, and Clustering) are analyzed. After that, we state the most commonly used algorithms that Mahout uses, giving a simple description of each one of them. Finally, we extensively study about the Clustering procedure in Mahout, which is going to concern us during this Thesis.

4.1. History of Mahout

The success of companies and individuals in the data age depends on how quickly and efficiently they turn vast amounts of data into actionable information. Whether it is for processing hundreds or thousands of personal e-mail messages per day or divining user intent from petabytes of weblogs, the need for tools that can organize and enhance data has never been greater. Therein lies the premise and the promise of the emerging field of machine learning and the usage of Mahout.

Mahout started in 2008 as a subproject of Apache's Lucene project, which was primarily concerned with content search and information retrieval technologies. Since there was much overlap between the techniques and algorithms used in the projects such as clustering and classification, on April of 2010, Mahout became a top-level Apache project in its own right.

More particularly, Mahout was developed as an Apache Java written open source project for building scalable machine learning libraries, with most algorithms built on top of Hadoop. More particularly, Mahout is a set of distributed data mining libraries that interface with an underlying distributed system, that of Hadoop, which implements the MapReduce framework, aimed to be used when the collection of data to be processed is very large, perhaps far too large for a single machine.

Thus, Mahout is used successfully for machine learning problems which involve very large collections of data. However, for small amounts of data, other libraries or products were faster and therefore better suited. So today, the Mahout library is suitable for applications that require scaling to large datasets, as with its machine learning

implemented algorithms it allows developers to create powerful and scalable clustering, classification data mining, and evolutionary programming. For this scalability, at a first place the algorithms were based on Apache Hadoop and the MapReduce paradigm, but later, on April 2014 the project decided to move to Apache Spark. [115],[118],[121],[122]

It is worth mentioning that Mahout is a work in progress, as the number of its developed and implemented algorithms are continuously growing. While Mahout's core algorithms for clustering, classification and recommender engines were implemented on top of Apache Hadoop using the MapReduce paradigm as mentioned above, it does not restrict contributions to Hadoop based implementations. Contributions that run on a single node or on a non-Hadoop cluster are also welcomed [113],[114],[116],[117],[120],[121].

It should be mentioned that there are many good frameworks that are user-friendly and fully equipped with equivalent algorithms to do the same tasks with Mahout, referring to clustering, classification, etc. For example, the R community is much bigger and in the Java world there is the RapidMiner and the Weka frameworks present on the scene for many years. The question that arises is why someone should use Mahout instead of the aforementioned frameworks. The truth is that all the previous frameworks are not meant to be designed for very large datasets.

The main power of Mahout lies in the fact that the algorithms are meant to be used in a Hadoop environment, which is a general framework that gives to an algorithm the ability to run in parallel on multiple machines, using the distributed MapReduce computing paradigm. Generally, the algorithms it implements fall under the broad umbrella of machine learning and data mining or collective intelligence.

Mahout also aims to [119]:

- Build and support a community of users and contributors such that the code outlives any particular contributor's involvement or any particular company or university's funding.
- Focus on real-world actual use cases, contrary skills and high-tech research.
- Provide a highly collaborative and reasonably well documented community, with high-quality articles and samples.
- Be contributed by programmers from all different computer science, data mining, and statistics disciplines.

4.2. Mahout Learning Techniques

Mahout incubates a number of techniques and algorithms, many still in development or in an experimental phase. Although it is, in theory, a project open to implementations of all kinds of machine learning techniques, it is in practice a project that focuses at the moment on three (3) basic key areas of machine learning. These are recommender engines, clustering, and classification. Although relatively young in the field of open source, Mahout has provided a number of features, especially in the Clustering, the Classification and the Recommender engines fields.

- *Recommender engines:* Recommender engines are the most immediately recognizable machine learning techniques in use today. Their main goal is to drive users to explore further offerings either from web sites or businesses. In particular, recommender engines provide recommendations to users based on a variety of patterns, and are helpful in guiding users to consider offerings that they might not otherwise be aware of, based on their specific user habits. For example, there are tons of services or web sites that attempt to recommend books or movies or articles based on past actions. They try to infer tastes and preferences and identify unknown items that are of interest.

  Some very popular web sites that make extensive use of the recommendation engines are Amazon, Netflix and Facebook. For instance Amazon, which is perhaps the most famous e-commerce site of deploying recommendations, recommends books and other items likely to be of interest of its users. Respectively, Facebook uses variants on recommender techniques to identify people who are most likely to be as-yet-unconnected friends [123].

- *Clustering:* Clustering is less apparent than recommender engines, but it turns up in equally well-known contexts. As its name implies, clustering techniques attempt to group a large number of things together into clusters that share some similarity. These clusters could be thought of as sets of items similar to each other in some ways, but dissimilar from the items belonging to other clusters. Clustering a group of data always involves three things: an algorithm, a notion of both similarity and dissimilarity, and a stopping condition. Generally speaking, clustering is a way to discover hierarchy and order in a large or hard-to-understand dataset, and in that way reveal interesting patterns or make the dataset easier to comprehend.

For example, Google News groups news articles by topic, using clustering techniques in order to present news grouped by logical story. Similarly, search engines, group their search results using various clustering techniques. Another general example are enterprises, which use clustering techniques so as to discover hidden groupings among users, to organize a large collection of documents sensibly, or to discover common usage patterns for a site based on logs.

- *Classification:* Classification techniques decide whether a thing is part of some type or category, or whether it has some attributes or not. Classification, like clustering, is ubiquitous, but it is even more behind the scenes. Characteristics of the classification function can include vocabulary, vocabulary weight (for example, according to the frequency) and voice parts. Often the systems that use classification techniques, learn by reviewing many instances of items in the categories in order to produce classification rules. The general idea of classification has many applications:

  For example, Yahoo! Mail decides whether incoming messages are spam or not, based on prior emails and spam reports from users, as well as on characteristics of the email itself. Similarly, Google's Picasa and other photo-management applications can decide when a region of an image contains a human face, using classification techniques. Apple's Genius feature in iTunes reportedly uses classification to classify songs into potential playlists for users [124].

Each of these techniques works best when provided with a large amount of good input data. In some cases, these techniques must not only work on large amounts of input, but also produce results quickly, making scalability a major issue. However, apart from the quality of the data, in order to be efficient and effective all these techniques, in each case they have to support and implement different kinds of algorithms, depending on the final output result that they expect [119],[120],[121],[122].

Table 4 contains a list of algorithms which have been developed in the terms of the Mahout's of Recommender Engines, Clustering and Classification techniques [119],[121],[122]. Mahout also includes some frequent itemset mining and machine learning algorithms that can be used, but those are not listed here.

TYPE OF CATEGORY	ALGORITHMS
RECCOMENDER ENGINES	• *Distributed Item-based Collaborative Filtering:* Estimates a user's preference for one item by looking at his/her preferences for similar items. • *Collaborative Filtering Using a Parallel Matrix Factorization:* Among a matrix of items that a user has not yet seen, predicts which items the user might prefer.
CLUSTERING	• *K-Means:* Partitions *n* observations into *k* clusters, where each observation belongs to the cluster with the nearest mean. • *Canopy:* Preprocesses the data before using K-Means. • *Fuzzy K-Means:* Discovers soft clusters where a particular point can belong to more than one cluster. • *Dirichlet Process:* Performs Bayesian mixture modeling. • *Latent Dirichlet Allocation:* Automatically and jointly clusters words into "topics" and documents into mixtures of topics. • *Mean Shift:* Finds clusters in 2-dimensional space, where the number of clusters is unknown. • *Minhash:* Quickly estimates similarity between 2 datasets. • *Spectral:* Clusters points using eigenvectors of matrices derived from the data.
CLASSIFICATION	• *Naïve Bayes:* Classifies objects into binary categories. • *Random Forests:* Operates by constructing a multitude of decision trees. • *Hidden Markov Models:* Is used for speech recognition, handwritten letter recognition or natural language processing. • *Logistic Regression:* Predicts the probability of occurrence of an event.

Table 4 - Mahout's Implemented Algorithms

In this Master thesis we examine the widely used field of Clustering, which has successfully adopted into Mahout via various well-known algorithms. The one that is going to concern us is that of the K-Means, which is going to be explained in full details, in the next Chapter.

4.3. Mahout Clustering

As mentioned before, one of the commonly used learning techniques of Mahout is the Clustering technique, which supports hundreds of different algorithms. Mahout, in turn, supports several of these developed algorithms, each with its own set of goals and criteria, its own benefits and defects. Regardless of the algorithm that someone is going to choose for clustering the data inside Mahout, the phases that are involved in the clustering process using Mahout (Figure 10) are always the same [121],[122],[125],[126]:

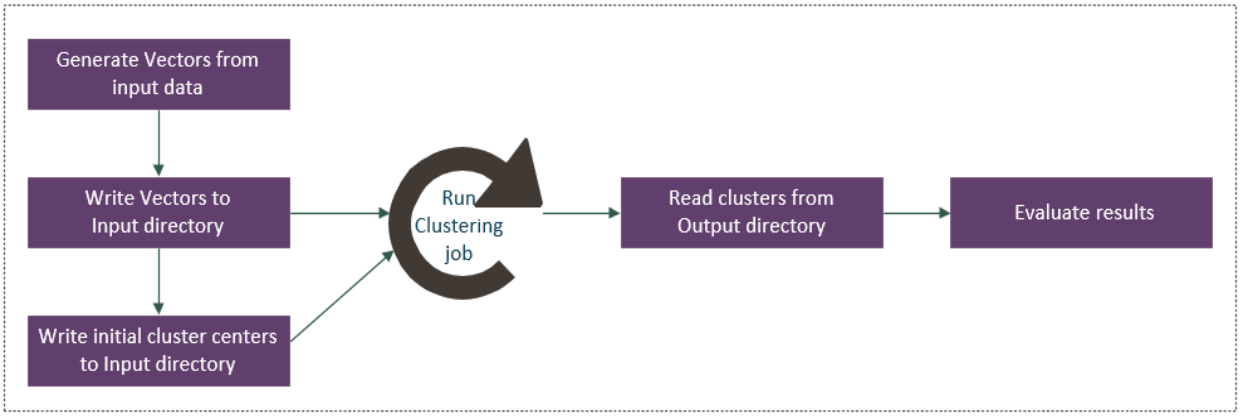


Figure 10 - Mahout's Clustering Process

1. Preprocessing of the input data.
2. Execution of the chosen clustering algorithm.
3. Evaluation of the output results.
4. Iteration on the procedure (if it is necessary).

In more details, in each step the following actions take place:

1. *Preprocessing of the input data:* Always clustering algorithms require data that is in a suitable format in order to be processed. Thus, in the first step of the preprocessing phase, the data have to be preprocessed. More specifically, if the data are in a text format it is mandatory to be converted firstly into a numeric representation, and secondly into a vector format (an array of weights that represent data). In all the other cases, when data is just numeric, it has only to be converted into vectors.

Mahout supports two (2) different kinds of vectors representations/classes, each one of which is optimized for different scenarios:

- *DenseVector*: Can be thought of as an array of doubles, whose size is the number of features in the data. It is called dense, due to the fact that all the entries in the array are pre-allocated regardless of whether the value is 0 or not. These vectors are commonly used when the values of the data are non-zero values.

- *SparseVector*: These vectors are mainly used for text-based problems and can be categorized into two different subgroups, the RandomAccessSparseVector and the SequentialAccessSparseVector.

  As for the *RandomAccessSparseVector*, it is implemented as a HashMap between an integer and a double, where only nonzero valued features are allocated, whilst the *SequentialAccessSparseVector* is implemented as two parallel arrays, one of integers and the other one of doubles. In contrast with the RandomAccessSparseVector, which is optimized for random access, SequentialAccessSparseVector is optimized for linear reading.

Depending on the data that someone wants to process and the algorithm that he is going to use for this processing, he will need to choose an appropriate implementation of vectors, in order to gain good performance. For example, if the algorithm does a lot of random insertions and updates of a vector's values, an implementation with fast random access like DenseVector or RandomAccessSparseVector is appropriate. On the other hand, for an algorithm like K-Means, which calculates the magnitudes of the vectors repeatedly, the SequentialAccessSparseVector implementation performs faster than the RandomAccessSparseVector.

After this step, when all the data have successfully converted into vectors, the second and final step of the phase takes places. In this step, all the vectors that have been created in the previous step have to be saved in a SequenceFile format, which will be used as an input for the Mahout's algorithm. As for the SequenceFile format, this is a binary format from the Hadoop library that encodes a series of key-value pairs, which in that case are the created vectors.

2. *Execution of the chosen clustering algorithm:* Once the input is ready, the chosen algorithm is able to begin its execution. More particularly, depending on the chosen clustering algorithm of Mahout, in order to begin clustering the data, some extra parameters need to be specified as an input of the algorithm. Some of the most commonly used parameters are the following:

   - A SequenceFile which contains the input vectors, which are necessary for the execution of the whole procedure.
   - A SequenceFile containing the initial cluster centers. (usually this SequenceFile is a part of the initial SequenceFile with the vectors)
   - A similarity measure to be used, choosing among the Euclidean distance measure, the Squared Euclidean distance measure, the Manhattan distance measure, the Cosine distance measure, the Tanimoto distance measure and the Weighted distance measure.
   - A convergence threshold, which is used in order to check whether in a particular iteration the centers of the clusters change beyond this threshold or not. In the case that the clusters do not change beyond this threshold, then no further iterations are done.
   - The maximum number of iterations to be done.

3. *Evaluation of the output results:* After running the chosen algorithm, in order to decide whether it was the proper one for clustering the given data or not, and whether the chosen input parameters were the appropriate ones or not, we have to analyze and evaluate the final results. Generally, analyzing the output of the clustering can be done with simple command-line tools or richer GUI-based visualizations. The first thing that someone has to do in order to evaluate the final results is to understand what the cluster looks like and the kind of features that are representative of each cluster. Moreover, it is necessary to see the distribution of the data points among different clusters, as for example someone will not want to end up in a situation where all k–1 clusters have one point each and the kth cluster has the rest of the points. Once someone understands what the clusters look like, he can focus on where he can improve the clustering. This means he needs to know how to tune the quality of input vectors, the distance measure, and all these input parameters that were mentioned above.

The primary tool in Mahout for inspecting the clustering output is the *ClusterDumper*. More specifically, ClusterDumper makes it easier to read the output of any clustering algorithm in Mahout, by providing a block of information according to the final results.

Apart from the ClusterDumper, Mahout provides another useful way for analyzing and evaluating the output results. This way refers to the metrics of the inter-cluster and the intra-cluster distances:

- *Inter-cluster:* Gives a nice view of what happened in after the execution of the clustering algorithm, by showing how near or far the resulting clusters ended up from each other.
- *Intra-cluster:* Gives the distance among the assigned data of a cluster, rather than the distance between two (2) different clusters, giving as a result a sense of how well the chosen distance measure was able to bring the data (items) together.

Chapter 5

Parallel Clustering with K-Means

In this chapter, we extensively study the use and the characteristics of the K-Means clustering algorithm, which is going to be implementing into two (2) different applications. More specifically, in the beginning of this chapter we state general information about K-Means, which is the chosen Data Mining Clustering algorithm that is going to be used in the terms of this Master Thesis. After this, are described in deep details the two (2) different implementations that have been constructed in this Thesis.

5.1. K-Means Algorithm

The K-Means algorithm is a prototype-based cluster analysis algorithm used as a partitioning method, and was developed by MacQueen in 1967. K-Means is one of the simplest and most widely used and studied clustering algorithm whose method is numerical, unsupervised, non-deterministic and iterative.

More particularly, K-Means is a simple algorithm for grouping objects, where all the objects need to be represented as a set of numerical features. Each object can be thought of as being represented by some feature vector in an n dimensional space, where n is the number of all the features used to describe the objects to cluster. Thus, the algorithm begins its execution by randomly choosing k points in that vector space and assigning each object to the center that it is closest to. Usually the distance measure is chosen by the user and determined by the learning task. After that, for each cluster a new center is computed, by averaging the feature vectors of all the objects that were assigned to it. The process of assigning objects and recalculating centers is repeated until the process converges, meaning that the process will be repeated until all the clusters converge. Thus, the algorithm can be proven to converge after a finite number of loops [121],[127],[128],[129].

In more details, the way that the K-Means algorithm works is quite simple, consisting of the following steps [130],[131],[132]:

1. The user specifies the number k of the clusters and the algorithm initiates k cluster centers, which represent the initial centroids.

2. Each point of the dataset is assigned to the closest cluster (there is always at least one point in each cluster), based upon a proximity measure. The Euclidean distance measure is the most common proximity measure, which calculates the distance (d) between two data points X (x_1, x_2,...x_n) and C (c_1, c_2,...c_n) with n attributes, using the following function:

$$\text{Distance d} = \sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2 + \ldots + (x_n - c_n)^2}$$

   Thus, according to the result of the distance measure, if the data point is closest to its own cluster, it is left where it is, whilst if the data point is not closest to its own cluster, it is moved into the closest cluster.

3. When all the data points have been assigned, each cluster center (centroid) is recomputed as the average of its including points. Mathematically, the values of the new centroids can be expressed as minimizing the sum of squared errors (SSE) of all data points of each individual cluster, using the function:

$$\text{SSE} = \sum_{i=1}^{k} \sum_{x_j \in C_i} ||x_j - \mu_i||2$$

   where C_i is the i^{th} cluster, j are the data points in the given cluster, X_j is a specific data object and μ_i is the centroid for the i^{th} cluster, which can be calculated with the following function:

$$\mu_i = \frac{1}{j_i} \sum_{x \in C_i} X$$

   where X is the data object vector (x_1, x_2,... x_n).

4. Steps 2 and 3 are repeated until the centroids no longer move. At this point the clusters are stable and the clustering process ends. Convergence may be defined differently depending upon the implementation, but it normally means that either none of the points change its cluster when steps 2 and 3 are repeated or that the changes do not make a material difference in the definition of the clusters.

Although it can be proved that the procedure will always terminate, K-Means does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centers and is computationally expensive, as it can be run multiple times to reduce this effect.

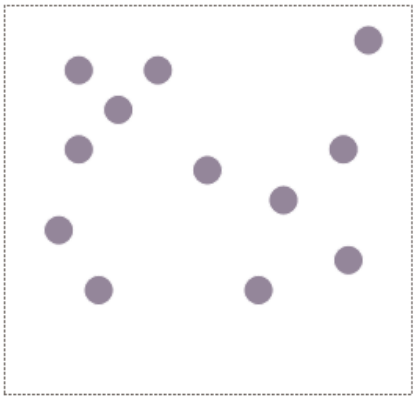
So, it becomes clear that K-Means is a very easy understandable and implementable algorithm that *benefits* its users in many perspectives:

- With a large number of variables, K-Means may be computationally faster than hierarchical clustering (if K is small).
- K-Means may produce tighter clusters than hierarchical clustering, especially if the clusters are globular.
- It works well with a variety of probability distributions.
- It is very simple, fast, robust and easy-to-understand.
- Gives best result when dataset are distinct or well separated from each other.

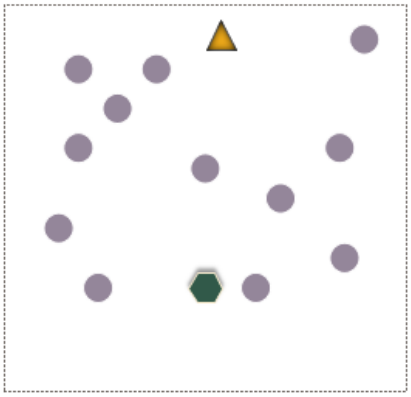
However, despite the wide popularity that the K-Means algorithm has, there are some significant *defects* that have led to the development of numerous alternative versions of K-Means during the past years:

- Difficulty in comparing quality of the clusters produced.
- As designed, the algorithm is not capable of determining the appropriate number of clusters and depends upon the user to identify this in advance.
- Algorithm fails for non-linear datasets.
- Different initial partitions can result in different (non-optimal) final clusters, and thus there might be a better optimal solution if the initial partitions change.
- It may take a high number of iterations to converge. Such number of iterations cannot be determined beforehand and may change.
- If there are two highly overlapping data, then K-Means will not be able to resolve that there are two clusters.
- Applicable only when mean is defined (e.g. fails for categorical data).
- Unable to handle noisy data and outliers.

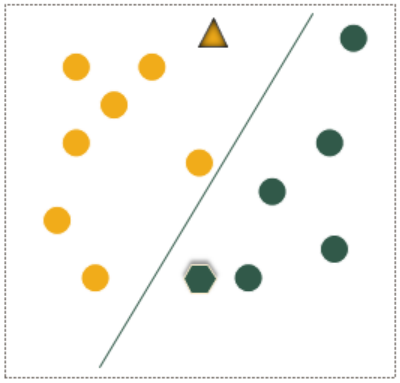
Finally, in order to fully understand the execution of the K-Means algorithm, a simple example is explained, where we initiate two (2) different centroids in a given dataset:



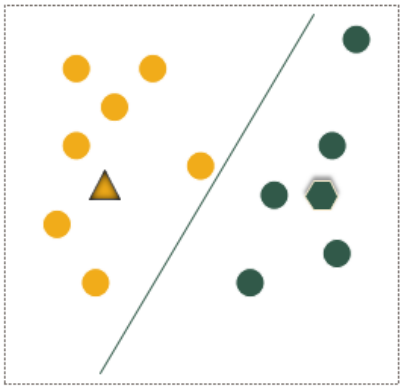
1. The input dataset with two dimensions.



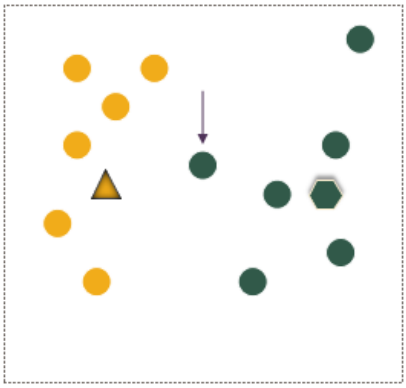
2. K-Means selects two random points as initial centroids.



3. All the data points are assigned to the nearest centroid to form a cluster.



4. For each cluster, it calculates the new centroids, by averaging the coordinates of the associated points of each cluster.



5. It assigns the data points to their new closest clusters.

After each iteration, the final configuration is fed back into the same loop until the centroids come to rest at their final positions.

5.2. K-Means Implementation on top of HDFS

In this section, it will be described in full details the implementation of the K-Means on top of HDFS, using Mahout's libraries. In simplest words, we are going to explain how the K-Means algorithm is performed by Mahout, and how Mahout is using the distributed computational and storing techniques of Hadoop, in order to achieve an efficient and successful clustering result.

The whole process of the K-Means clustering using Mahout and Hadoop is quite simple. In short, at the beginning of the algorithm, apart from all the input parameters that the algorithm's implementation needs in order to begin its execution, it accepts as an input two different sequence files, which are stored in the Hadoop Distributed File System (HDFS). The first one contains the whole dataset converted into a vector format, and the other one contains the initial clusters, according to the chosen number of clusters. Then, having received as an input these two files, the algorithm begins with its first iteration, where in each one of them, it stores the clusters' output in HDFS in a directory named as clusters-x, where x equals with the iteration number. During this process, the MapReduce paradigm is being used, which executes the same number of Map and Reduce phases in order to preprocess in parallel the data, whose results are stored in the aforementioned folder (clusters-x). When the algorithm finishes its iterations, the final results of the MapReduce jobs are stored again in HDFS, in another folder named as clusters-x-final (x is the iteration number).

At this point, it should be mentioned that the algorithm stops running either when the maximum number of iterations has been reached or the created clusters have converged, according to a given threshold.

In Figures 11 - 15, it is given a full image of how the K-Means algorithm works on top of HDFS, using Mahout's libraries. For each one of the Figures, it will be given an explanation of how the algorithm works, by explaining the exact steps that it follows in order to cluster the whole dataset.

❖ Preprocessing Phase:

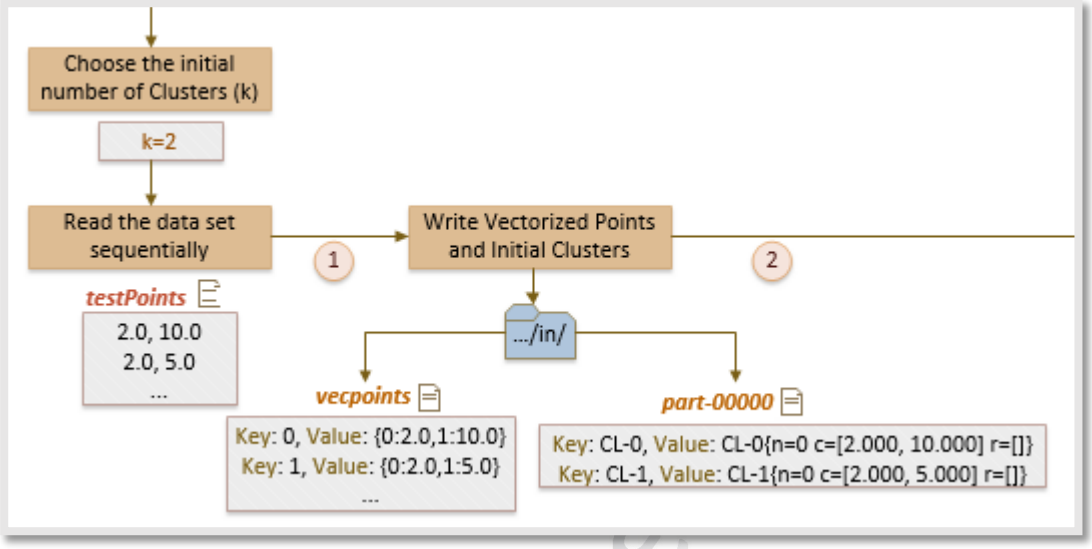


Figure 11 - Preprocessing Phase of K-Means using Mahout and HDFS

The preprocessing step, is the first step of the whole procedure. In this phase, at first place the user has to choose the number of the clusters that he wants to use the K-Means algorithm during the clustering process.

Example: The user chooses 2 clusters (k=2).

1 After this, the program takes over for the completion of the whole procedure. In particular, it copies the input text file from the user's local folder to the HDFS home directory, splitting it into blocks of the 64MB. Afterwards, it reads from the HDFS sequentially the input text file which contains all the data points that need to be processed. It should be mentioned, that the algorithm has constructed to read only numerical non-noisy data, which are separated with commas (",").

Example: We have an input file called "testPoints" with the following format:

```
2.0,10.0
2.0,5.0
8.0,4.0
5.0,8.0
7.0,5.0
...
```

After the successful reading of the input file, the program creates in the HDFS the input folder "in" that is needed in order to be stored all the created files that are needed as an input for the execution of the K-Means algorithm. More particularly, two sequence files are written inside this folder. The first one is a sequence file called "vecpoints", which contains all the input data in a vectorised format, written in key-value pairs. The second one is a sequence file called "part-00000", and contains all the created initial centroids, which always are the first different points of the input file, written in key-value pairs, as well.

Example: The folder "in" is created in the HDFS, with the following sequence files:

"vecpoints": contains the vectorised input data stored in key-value pairs, where key is an auto increment id and value the vectorised value (coordinates) of each data point.

Key: 0, Value: {0:2.0,1:10.0}
Key: 1, Value: {0:2.0,1:5.0}
Key: 2, Value: {0:8.0,1:4.0}
…

"part-00000": contains the two (2) created initial centroids, according to the number k that the user has chosen, having as a key the identifier of each centroid and as a value the cluster format of the each centroid, where *n* is the total number of observations (belonging points) of the cluster, *c* is the coordinates of the center and *r* is the radius of the centroid.

Key: CL-0, Value: CL-0{n=0 c=[2.000, 10.000] r=[]}
Key: CL-1, Value: CL-1{n=0 c=[2.000, 5.000] r=[]}

❖ Execution of the K-Means algorithm

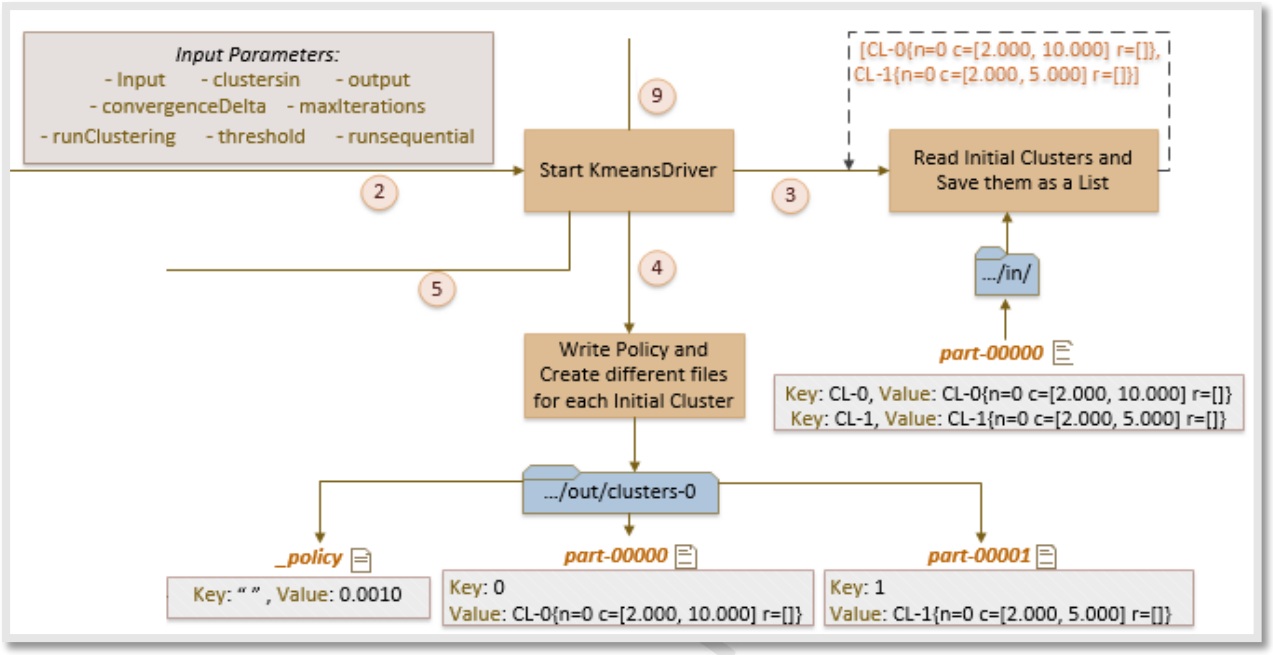


Figure 12 - Execution Phase of K-Means using Mahout and HDFS

2 After the preprocessing phase, the execution of the K-Means algorithm begins. In order to begin its clustering procedure, K-Means needs some input parameters that is going to be used during the data's clustering. Most of these parameters have already defined (section 4.3.), as they are common for almost all the clustering algorithms that have been implemented in Mahout. More specifically, the K-Means needs eight (8) different input parameters, which are the following:

- *input*: The directory pathname for the vectorised input data points.
- *clustersIn*: The directory pathname for the initial created centroids.
- *output*: The directory pathname for the output results of the algorithm.
- *convergenceDelta*: The convergence delta value that is used by the algorithm in each iteration in order to decide whether the clusters converge or not.
- *maxIterations*: The maximum number of iterations that the algorithm can execute.
- *runClustering*: Indicates whether after the completion of the algorithm the input data need to clustered or not. If this parameter equals with true, then the program has to cluster the input data points after the iterations are completed. Otherwise, K-Means does not have to cluster the input data after the iterations are completed, and the procedure of the clustering completes after the convergence check.
- *clusterClassificationThreshold*: A clustering strictness/outlier removal parameter.
- *runSequential*: The parameter that decides whether the algorithm has to run sequentially (runSequential = true) or in parallel (runSequential = false).

3 Then, the program reads the "part-00000" sequence file with the initial centroids and saves them in a list, which contains the detailed value information for each cluster.

Example: Reading the "part-00000" sequence file with the two (2) initial centroids, the program saves them into the list:

[CL-0{n=0 c=[2.000, 10.000] r=[]}, CL-1{n=0 c=[2.000, 5.000] r=[]}]

4 After this step, the program creates in the HDFS an output folder "out" that is needed in order to be stored all the output created files of the algorithm. Inside this folder, is created another folder with the name "clusters-0", whose name states the algorithm's iteration number, which in this case is 0, as the algorithm has not yet run any iterations. Finally, inside this sub-folder (clusters-0) are created several different sequence files, each one of these for its own purpose. The first one is a sequence file called "_policy", which contains the value of the convergence delta parameter. Apart from this, for each initial centroid that the sequence file "part-00000" contains, it creates a single separate sequence file, with the name "part-…", which contains the individual key-value information for each centroid.

Example: The folders "out" and "clusters-0" are created in HDFS, with the following sequence files:

"_policy": contains the convergence delta for the K-Means algorithm:

Key: "", Value: 0.0010

As we have two (2) different centroids, there will be created two (2) different part-... files, the part-00000 and the part-00001.

"part-00000": contains the cluster info for the first initial centroid:

Key: 0, Value: CL-0{n=0 c=[2.000, 10.000] r=[]}

"part-00001": contains the cluster info for the second initial centroid:

Key: 1, Value: CL-1{n=0 c=[2.000, 5.000] r=[]}

❖ Execution of the Map Phase

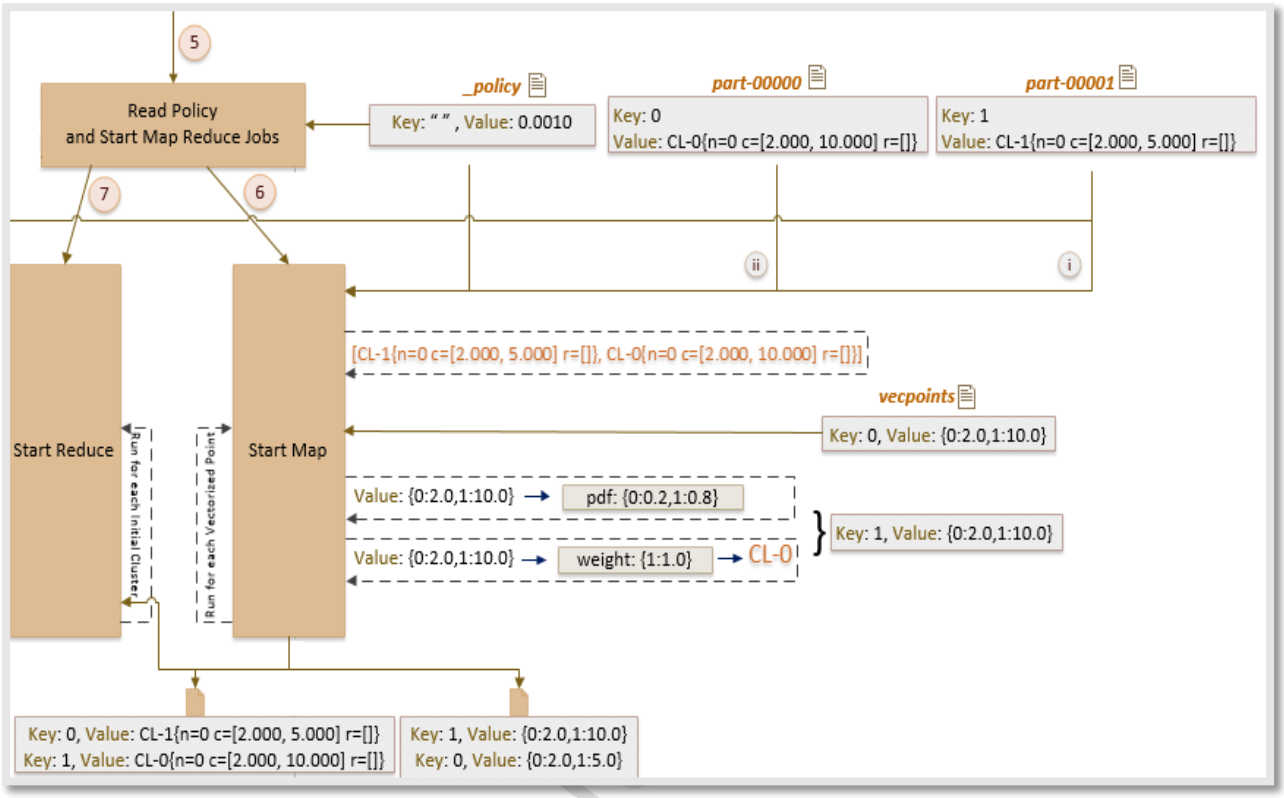


Figure 13 - Map Phase of K-Means using Mahout and HDFS

5 Having completed successfully all the previous steps, it is time for the execution of the MapReduce jobs. Before executing the Map phase, the program reads the "_policy" sequence file that was created in the previous step.

6 In this step, the jobs of the Map phase take place. In particular, the program reads either the "part-..." sequence files in a descending order only for the first iteration, or the "part-r-00000" sequence file for the rest of the iterations and once again the "_policy" sequence file of the current iteration's folder. By reading these sequence files it keeps the cluster value information of each cluster and saves them in a list.

Example: In the first iteration, the program reads the "part-00001" and "part-00000" sequence files (in descending order) with their two (2) corresponding initial centroids, and saves them into a the list:

[CL-1{n=0 c=[2.000, 5.000] r=[]}, CL-0{n=0 c=[2.000, 10.000] r=[]}]

Then, the program reads the "vecpoints" sequence file with the vectorised input data and iterates over it, by reading one by one the key-value pairs that it contains.

> Example: The first key-value pair that is read from the "vecpoints" sequence file is:
>
> Key: 0, Value: {0:2.0,1:10.0}

Thus, for each vectorised data point that it reads, it implements the following procedure:

- It calculates the pdf (probability density function -probability of a vectorised point to belong to a cluster-) from each centroid that exists in the above list, and saves all the calculated probabilities into a list. In order to calculate the pdf, it uses an exact formula, which will be explained using the following example.

  > Example: During the first iteration over the input file, the algorithm reads the vectorised point {0:2.0,1:10.0}. Comparing this with the center value (c) of the first centroid of the list {0:2.0,1:5.0}, is calculated the pdf between this given point and the centroid, using the formula:
  >
  > $$pdf = 1 / (1 + sqrt((2.0 - 2.0)^2 + (10.0 - 5.0)^2)) = 0.16666$$
  >
  > Afterwards, we do exactly the same calculations for the rest of the centroids, saving the results into a list:
  >
  > [0.16666, 0.83333]

- Then, it converts the list with the calculated pdfs into a vector format, and using this vector, it calculates the final pdf value of each cluster, it adds it to a list, and finally converts this list with the final pdfs into a vector format. The formula that is used in order to calculate the final value of the pdf will be explained below with an example:

  > Example: Using the calculated pdf value between the vectorised point {0:2.0,1:10.0} and the center value (c) of the first centroid of the list {0:2.0,1:5.0}, now is calculated the final value of the pdf, using the formula:
  >
  > final pdf = (1 / (0.16666 + 0.83333)) * 0.16666 = 0.16666

Afterwards, we do exactly the same calculations for the rest of the centroids, saving the results into a list, which finally is converted into a vector format:

{0:0.16666,1:0.83333}

- After this step, it returns the vectorised format of its weight, which indicates the position – index of the maximum calculated probability (pdf) value.

Example: From the previous example we can notice that the index-position with the greater value is the value with index one (1:), so the final weight of the data point {0:2.0,1:10.0} is {1:1.0}, which means that this point ({0:2.0,1:10.0}) from now on belongs to the cluster with identifier 0 (CL-0).

- Finally, the vectorised point accompanied with its calculated weight are buffered in memory in a key-value pair format, having as a key the calculated weight and as a value the vectorised point. At the same time, it writes into the memory another sequence file with the current values of the centroids, finalizing all the tasks that the Map phase has to do. Thus, at this step, the Map phase prepared its output for the Reduce phase, which are the key-value pairs of the data points and the clusters that they belong to.

Example: The vectorised points and their weights are buffered in memory, containing the following key-value pairs:

Key: 1, Value: {0:2.0,1:10.0}
Key: 0, Value: {0:2.0,1:5.0}
…

As for the sequence file with the current values of the centroids, this contains the following key-value pairs:

Key: 0, Value: CL-1{n=0 c=[2.000, 5.000] r=[]}
Key: 1, Value: CL-0{n=0 c=[2.000, 10.000] r=[]}

❖ Execution of the Reduce Phase

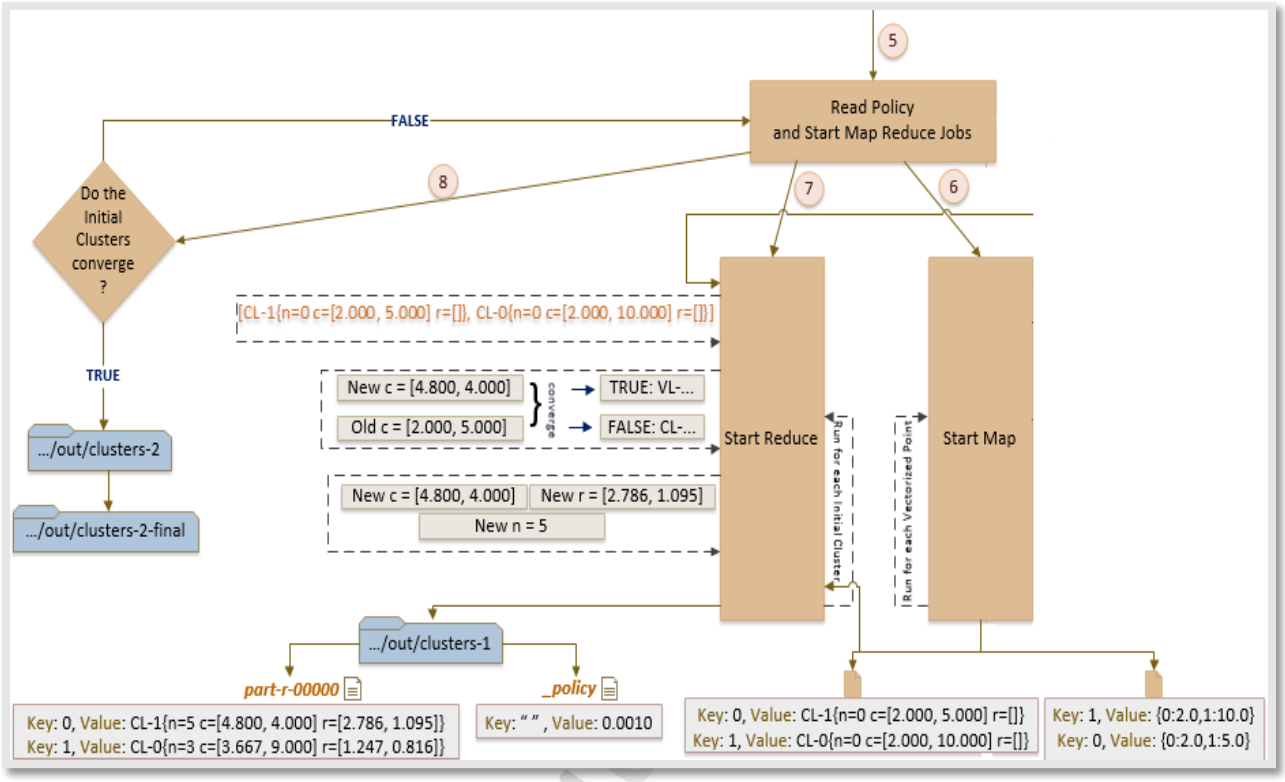


Figure 14 - Reduce Phase of K-Means using Mahout and HDFS

7 Having completed successfully the Map phase, the jobs of the Reduce phase take place. In particular, the program reads either the "part-..." sequence files in a descending order for the first iteration or the "part-r-00000" for the rest of the iterations, and once again the "_policy" sequence file of the current iteration's folder. By reading these sequence files it keeps the information of each cluster and saves them in a list.

Example: In the first iteration, the program reads the "part-00001" and "part-00000" sequence files (in descending order), and saves their corresponding centroids into a list:

[CL-1{n=0 c=[2.000, 5.000] r=[]}, CL-0{n=0 c=[2.000, 10.000] r=[]}]

Then, the program reads from the memory the vectorised input data and their corresponding weights (the file that was exported from the Map phase), and according to these values it calculates the new center for each centroid. More particularly, in order to calculate the new centers for each centroid, it finds all the input points with the same weight value, and calculates the average of their coordinates.

Example: If the data points {0:2.0,1:10.0} and {0:2.0,1:5.0} had as a weight the value one (1:) which would mean that they belong to the cluster 0 (CL-0), then in order to calculate cluster's 0 new centroid, we would use the following formula:

$$c = [(2.0+2.0)/2, (10.0 + 5.0)/2] => c = [2.000, 7.500]$$

Afterwards, it uses the values of the current iteration center centroids (new centroids) and the previous iteration's center centroids (old centroids), in order to compare them with the value of the convergence delta. If the number that it calculated is <= of the convergence delta, then the centroids converge, and as a result, the new centroid takes as an identifier the value VL-… instead of CL-… . On the other hand, if the two compared centroids do not converge, then the new centroid continues having as an identifier the value CL-… . In particular the way that the convergence is calculated, is described in the following example:

Example: In the first iteration, cluster 0 (CL-0) had as an initial center the values [2.000, 10.000], while after the calculation of its new center which was made in the previous step, it had the value [2.000, 7.500]. Thus, in order to decide whether the cluster 0 has converged or not, we use the following formula:

$$converge = sqrt((2.0^2 + 7.5^2) – 2 * (2.0 * 2.0 + 10.0 * 7.5) + (2.0^2 + 10.0^2)) = 2.5$$

- if converge > convergence delta, then the new cluster does not converge with the old one, and the algorithm returns a "false" boolean value

- if converge <= convergence delta, then the new cluster converges with the old one, and the algorithm returns a "true" boolean value

After the completion of the above steps, the new total number of observations, the new center and the new radius are calculated for each centroid, which finally are written in the HDFS in the output folder. In particular, a new sub folder is created, called "clusters-1", where "1" refers to the number of the iteration's folder. Thus, inside this folder is created a new "_policy" sequence file and another sequence file called "part-r-00000", which contains all the values of the new calculated centroids, written in key-value pairs.

Example: In the completion of the first iteration, the folder "clusters-1" is created, containing the following sequence files:

"_policy": contains the convergence delta for the K-Means algorithm:

Key: "", Value: 0.0010

"part-r-00000": contains the cluster information about the new calculated centroids

Key: 0, Value: CL-1{n=5 c=[7.000, 4.333] r=[0.816, 0.471]}
Key: 1, Value: CL-0{n=3 c=[1.500, 3.500] r=[0.500, 1.500]}

Thus, at this moment the jobs of the Reduce phase are completed, giving as an output of the algorithm, the new calculated centroids.

8 After the successful creation of the sequence file "part-r-00000", a final basic step takes place, in which it is checked whether the new centroids have converged or not with the old ones. More specifically, this is only done by checking whether all the identifiers of the centroids have the value VL-… or not. As a consequence, in this step the algorithm checks whether the previously returned convergence boolean value has either the value "true" of "false". If all the centroids have as an identifier the value VL-… (value = "true"), then that means all the new centroids converge with the old ones, and the algorithm will not iterate again and it will rename the clusters-... folder to the folder clusters-...-final. Otherwise, the algorithm continues running, by iterating again from the step 6 and so on. However, it should me mentioned, that if at least one value has the identifier CL-… and all the other values have the identifier VL-…, then the algorithm will not stop, but it will continue running, until all the value clusters have converged (identifier VL-…).

Example: The first iteration's new centroids that were calculated have the identifier CL-1 and CL-0, which means that the new centroids did not converge with the old ones, and the algorithm continues its execution.

❖ Completion of the algorithm

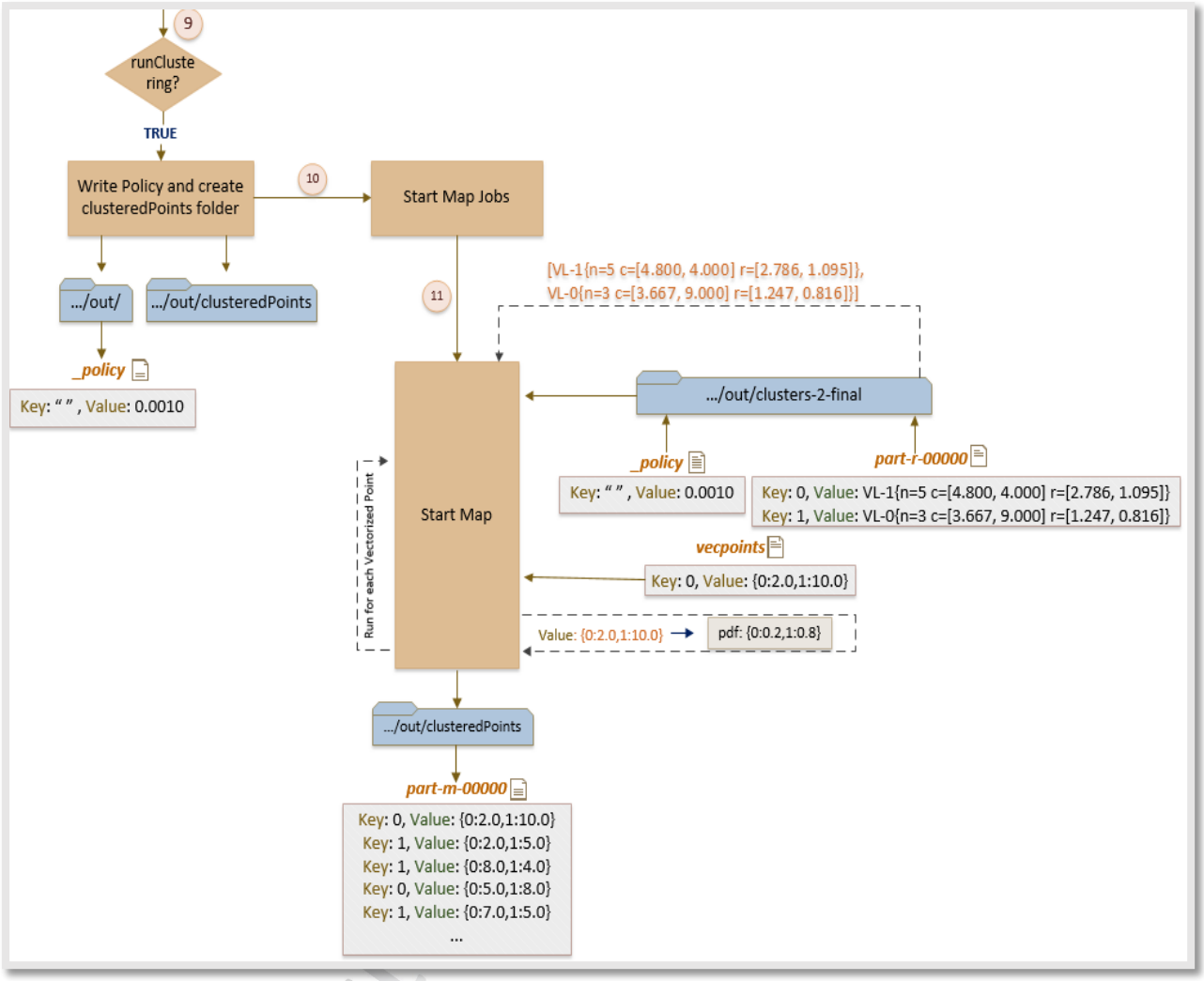


Figure 15 - Completion of K-Means using Mahout and HDFS

At this part of the program two different actions can be done, depending on the chosen value of the input parameter runClustering of the algorithm:

- If *runClustering* equals with *true*, then the algorithm has to cluster the input data points after the iterations are completed, following a procedure that is going to be explained below.
- If *runClustering* equals with *false*, then the algorithm does not have to cluster the input data points after the iterations are completed, and the procedure of the clustering completes after the convergence check.

So, in the first case (runClustering = true) the procedure that the algorithm follows is:

9 First of all, it creates in the output folder "out" of the HDFS both the "_policy" sequence file and the sub folder "clusteredPoints".

10 After this, the jobs of the Map phase take place. In particular, the program reads the "part-r-00000" sequence file and the "_policy" sequence file of the final iteration's folder "clusters-...-final". By reading the "part-r-00000" sequence file it keeps the cluster value information of each cluster and saves them into a list.

Example: By reading the "part-r-00000" sequence file with the two (2) corresponding converged centroids, the program saves the list:

VL-1{n=2 c=[1.500, 3.500] r=[0.500, 1.500]}, VL-0{n=3 c=[3.667, 9.000] r=[1.247, 0.816]}]

Then, the program reads the "vecpoints" sequence file with the vectorised input data and iterates over it, by reading one by one the key-value pairs that it contains.

Example: The first key-value pair that is read from the "vecpoints" sequence file is:

Key: 0, Value: {0:2.0,1:10.0}

Thus, for each vectorised data point that it reads, it implements the following procedure:

- It calculates the pdf (probability density function) from each centroid of the above list, and saves all the calculated probabilities into a list. In order to calculate the pdf, it uses an exact formula, which will be explained using the following example.

  Example: During the first iteration over the input file, the algorithm reads the vectorised point {0:2.0,1:10.0}. Comparing this with the center value (c) of the first centroid of the list {0:2.0,1:5.0}, is calculated the pdf between this given point and the centroid, using the formula:

  $$pdf = 1 / (1 + sqrt((2.0 - 2.0)^2 + (10.0 - 5.0)^2)) = 0.16666$$

  Afterwards, we do exactly the same calculations for the rest of the centroids, saving the results into a list:

  [0.16666, 0.83333]

- Then, it converts the list with the calculated pdfs into a vector format, and using this vector, it calculates the final pdf value of each cluster, it adds it to a list, and finally converts this list with the final pdfs into a vector format. The formula that is used in order to calculate the final value of the pdf will be explained below with an example:

> Example: Using the calculated pdf value between the vectorised point {0:2.0,1:10.0} and the center value (c) of the first centroid of the list {0:2.0,1:5.0}, now is calculated the final value of the pdf, using the formula:
>
> final pdf = (1 / (0.16666 + 0.83333)) * 0.16666 = 0.16666
>
> Afterwards, we do exactly the same calculations for the rest of the centroids, saving the results into a list, which finally is converted into a vector format:
>
> {0:0.16666,1:0.83333}

- Finally, it saves the vectorised point accompanied with its calculated weight into the "part-m-00000" sequence file which is created into the folder "clusteredPoints" in HDFS, having as a key the id of the cluster that each point belongs to and as a value the vectorised data point. This is the final output of the Map phase, and generally of the K-Means algorithm, as at this place there will be no Reduce phase, as there is no need for its existence.

> Example: The "part-m-00000" sequence file with the vectorised points and their weights, contains the key-value pairs:
>
> Key: 1, Value: {0:2.0,1:10.0}
> Key: 0, Value: {0:2.0,1:5.0}
>
> …

Thus, the final results of all this procedure is the "part-m-00000" output sequence file of the single Map phase, which contains all the input data points grouped into clusters and the "part-r-00000" output sequence file of the Reduce phase, which contains the final clusters. For that reason, the final output result that is provided to the user answers to the question of which point belongs to which cluster (e.g. {0:2.0,1:10.0} belongs to Cluster 0) and which are the final values of the created clusters (e.g. VL-1{n=2 c=[1.500, 3.500] r=[0.500, 1.500]}).

5.3. K-Means Implementation on top of an OLAP database

In this section, it will be described in full details the implementation of the K-Means algorithm on top of an OLAP database, using Mahout's machine learning libraries. More specifically, we are going to explain how the K-Means algorithm works inside Mahout, but instead of using as a storage the HDFS accompanied with the MapReduce paradigm, it will use an OLAP database in combination with a new developed distributed key-value pair concept, which was created in the terms of this Master Thesis.

In general, the implementation of K-Means on top of an OLAP database, has many differences in comparison with the implementation that it was explained in section 5.2. More particularly, the two main differences between these two implementations are mentioned below:

- In the new implementation, the idea of HDFS does not exist anymore, and for that reason, all the aforementioned files that were created and stored in different folders and sub-folders in HDFS, are now stored in different tables in the OLAP database.

- The MapReduce paradigm has been fully replaced by a different distributed key-value pair concept. In this new concept there are three (3) different phases in the whole procedure of the data's processing (Map-Reduce-FinalReduce), which are based upon the creation of multiple threads that read and write in parallel different data on OLAP's database tables, and do the same jobs without interrupting each other. In particular, at the beginning of this procedure, in the Map phase there are created numerous threads running in parallel different Map jobs. Each one of these threads, is assigned to process a specific fragment of data without having to communicate with the other existing threads, and produce as a result different key-value pairs. Then, in the Reduce phase each one of these threads continues its corresponding Reduce job, processing its Map's output key-value pairs, without waiting for the other threads to finish their Map tasks. Finally, in the FinalReduce phase, all the output key-value pairs of the Reduce phase of each thread are merged and the final key-value pairs are created.

More details about how the K-Means implementation works on top of an OLAP database using Mahout's libraries, will be given and explained in Figures 16 - 20. In each Figure, it will be shown how the algorithm works in comparison with the old one implementation, stating again the exact steps that the algorithm follows in order to process the data.

❖ Preprocessing Phase:

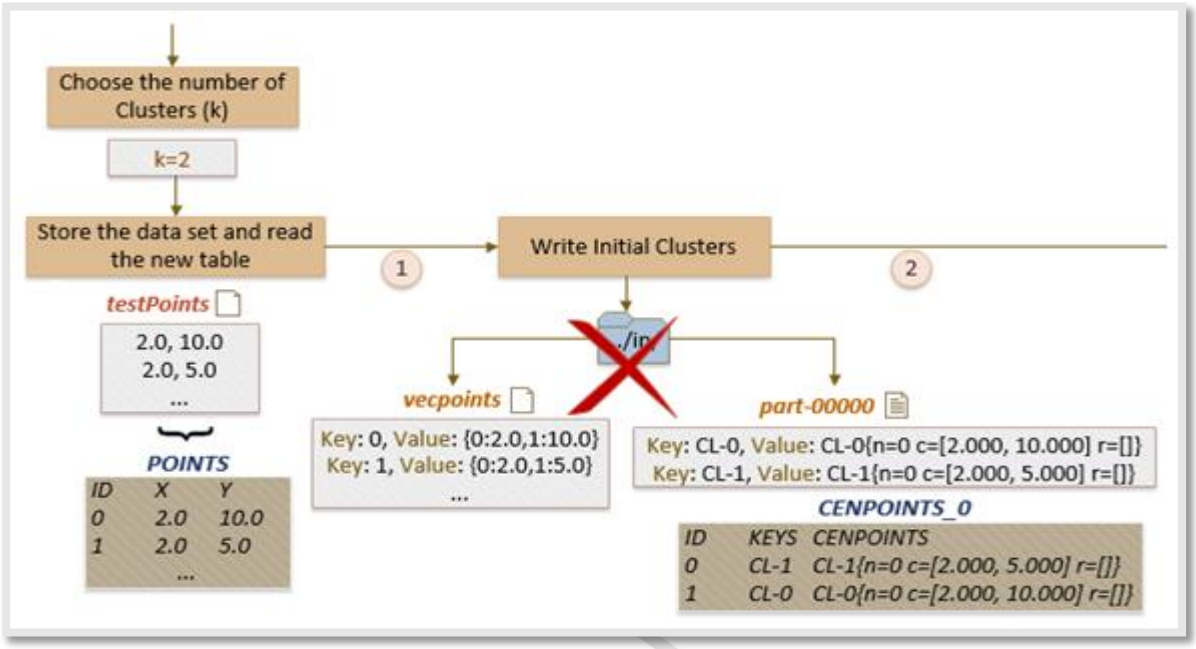


Figure 16 - Preprocessing Phase of K-Means using Mahout and an OLAP database

The preprocessing step, is the first step of the whole procedure. In this phase, at a first place the user has to choose the number of the clusters that he wants to use the K-Means algorithm during the clustering process.

Example: The user chooses 2 clusters (k=2).

1 After this, the program takes over for the completion of the whole procedure. In particular, it reads the input text file from the user's local folder and stores it in the table "POINTS" of the OLAP database. It should be mentioned, that the algorithm has constructed to read only numerical non-noisy data, separated with commas (",").

Example: The "testPoints" input file, inserts its data into the table "POINTS":

```
2.0,10.0
2.0,5.0
8.0,4.0
5.0,8.0
7.0,5.0
```

ID	COLUMN_0	COLUMN_1
0	2.0	10.0
1	2.0	5.0
2	8.0	4.0
3	5.0	8.0
4	7.0	5.0

After the successful insert of the input file, the program creates another table called "CENPOINTS_0", in which are stored all the initial centroids, which always are the first different points of the input file.

Example: The table "CENPOINTS_0" is created, containing the first two (2) different values of the table "POINTS" into a cluster format:

ID	KEYS	CENPOINTS
0	CL-1	CL-1{n=0 c=[2.000, 5.000] r=[]}
1	CL-0	CL-0{n=0 c=[2.000, 10.000] r=[]}

❖ Execution of the K-Means algorithm

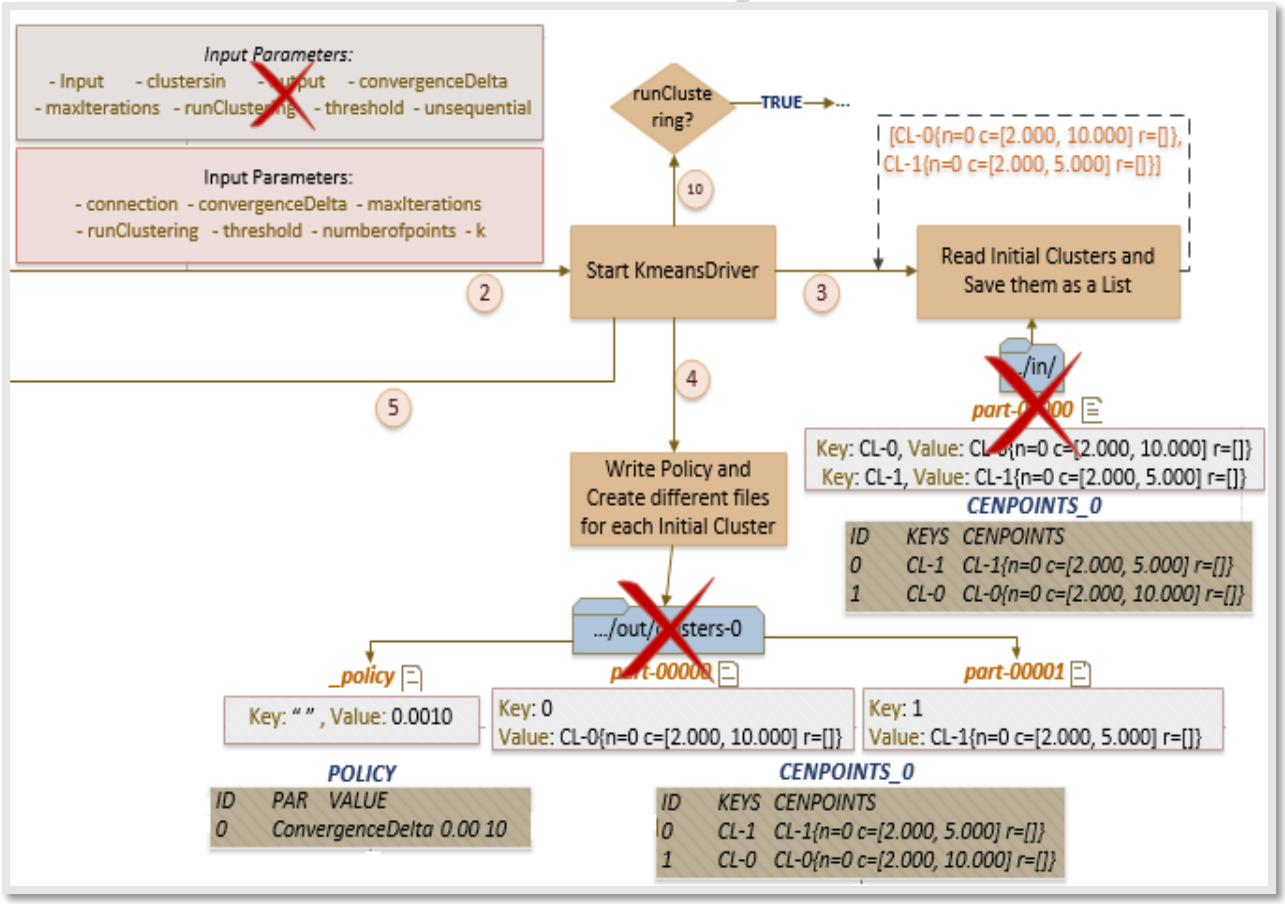


Figure 17 - Execution Phase of K-Means using Mahout and an OLAP database

2 After the preprocessing phase, the execution of the K-Means algorithm begins. In order to begin its clustering procedure, K-Means needs some input parameters that is going to be used during the data's clustering. More specifically, the K-Means needs seven (7) different input parameters, which are the following:

- *connection*: The connection to the created OLAP database.
- *convergenceDelta*: The convergence delta value that is used by the algorithm in each iteration in order to decide whether the centroids converge or not.
- *maxIterations*: The maximum number of iterations that the algorithm can execute.
- *runClustering*: Indicates whether after the completion of the algorithm the input data need to clustered or not. If this parameter equals with true, then the program has to cluster the input data points after the iterations are completed. Otherwise, it does not have to cluster the input data points after the iterations are completed, and the procedure of the clustering completes after the convergence check.
- *clusterClassificationThreshold*: A clustering strictness/outlier removal parameter.
- *numberofpoints*: The total number of the inserted records of the table "POINTS".
- *k*: The number of the clusters that have created.

3 Then, the program reads (backwards) the values of the CENPOINTS field of the table "CENPOINTS_0", and saves them into a list.

Example: It reads the values CL-1{n=0 c=[2.000, 5.000] r=[]} and CL-0{n=0 c=[2.000, 10.000] r=[]} of the table "CENPOINTS_0", and saves them into a list:

ID	KEYS	CENPOINTS
0	CL-1	CL-1{n=0 c=[2.000, 5.000] r=[]}
1	CL-0	CL-0{n=0 c=[2.000, 10.000] r=[]}

[CL-0{n=0 c=[2.000, 10.000] r=[]}, CL-1{n=0 c=[2.000, 5.000] r=[]}]

4 After this, it creates a table called "POLICY", which contains all the information about the K-Means policy, which in this case is only the convergence delta.

Example: The table "POLICY" is created, containing the following values:

ID	PAR	VALUE
0	ConvergenceDelta	0.0010

❖ Execution of the Map Phase

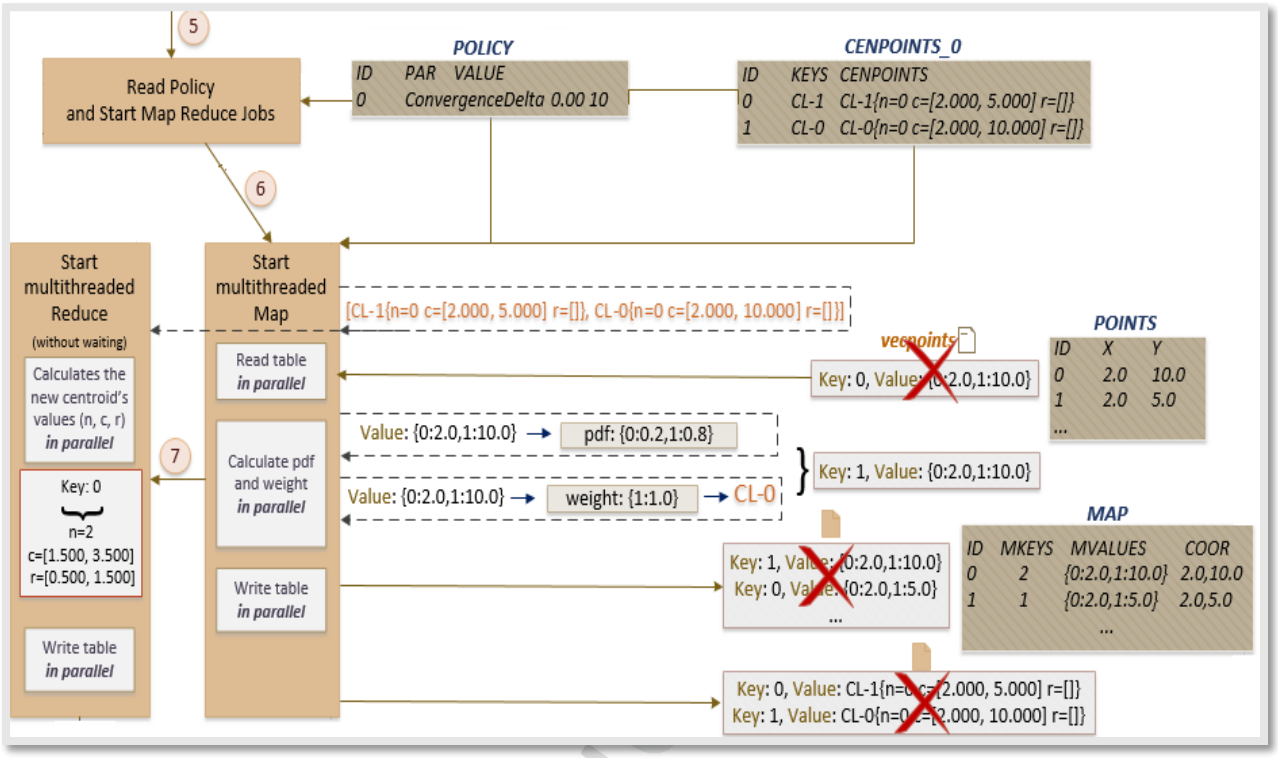


Figure 18 - Map Phase of K-Means using Mahout and an OLAP database

5 Having successfully completed all the previous steps, it is time for the execution of the MapReduce jobs. Before executing the Map phase, the program reads the convergence delta value of the "POLICY" table that was created in the previous step.

6 In this step, the jobs of the Map phase take place. However, in this part of the program, it is not used the MapReduce paradigm of Hadoop, but another multithreaded implementation constructed by us. In particular, according to the number of the data points that are stored in the table "POINTS", there are created different threads according to the size of the table, whose purpose is to split the jobs of the Map, by running independently in parallel.

Example: If the table "POINTS" had 1000 rows (1000 different data points), then the number of the threads that were going to be created would be 10, assuming that we assign to each thread 100 different data points.

So, after the successful creation of the threads, each one of them reads the cluster information for each existing centroid from the table "CENPOINTS_0", saves these information into a list, and reads the convergence delta value of the "POLICY" table.

Example: Each thread reads the values CL-1{n=0 c=[2.000, 5.000] r=[]} and CL-0{n=0 c=[2.000, 10.000] r=[]} of the table "CENPOINTS_0", saves them into its single list, and reads the value 0.0010 from the table "POLICY":

"CENPOINTS_0"
(for the first iteration)

ID	KEYS	CENPOINTS
0	CL-1	CL-1{n=0 c=[2.000, 5.000] r=[]}
1	CL-0	CL-0{n=0 c=[2.000, 10.000] r=[]}

"CENPOINTS_0"
(for the rest of the iterations)

ID	KEYS	CENPOINTS
0	0	CL-1{n=0 c=[2.000, 5.000] r=[]}
1	1	CL-0{n=0 c=[2.000, 10.000] r=[]}

[CL-1{n=0 c=[2.000, 5.000] r=[]}, CL-0{n=0 c=[2.000, 10.000] r=[]}]

POLICY

ID	PAR	VALUE
0	ConvergenceDelta	0.0010

Then, the program assigns to each thread to read a specific part of the table "POINTS". More specifically, each thread is assigned to read in parallel the data point values between two (2) different IDs of the table "POINTS", using as a parameter a min and a max value.

Example: Assume that the table "POINTS" has the following five (5) records, and that each thread is assigned to read only three (3) records. Thus, as we have five (5) records, two (2) different threads are going to be created, each one of them reading three (3) different records, according to their IDs. For that reason, the first thread reads all the records with IDs 0-2 and the second one the records with the IDs 3-4.

ID	COLUMN_0	COLUMN_1
0	2.0	10.0
1	2.0	5.0
2	8.0	4.0
3	5.0	8.0
4	7.0	5.0

Thus, for each data point that each thread reads, it implements the following procedure:

- It calculates the pdf (probability density function -probability of a vectorised point to belong to a cluster-) from each centroid that exists in the above list, and saves all the calculated probabilities into a list. In order to calculate the pdf, it uses an exact formula, which will be explained using the following example.

  Example: During the first iteration over the input file, the algorithm reads the vectorised point {0:2.0,1:10.0}. Comparing this with the center value (c) of the first centroid of the list {0:2.0,1:5.0}, is calculated the pdf between this given point and the centroid, using the formula:

  $$pdf = 1 / (1 + sqrt((2.0 - 2.0)^2 + (10.0 - 5.0)^2)) = 0.16666$$

  Afterwards, we do exactly the same calculations for the rest of the centroids, saving the results into a list:

  [0.16666, 0.83333]

- Then, it converts the list with the calculated pdfs into a vector format, and using this vector, it calculates the final pdf value of each cluster, it adds it to a list, and finally converts this list with the final pdfs into a vector format. The formula that is used in order to calculate the final value of the pdf will be explained below with an example:

  Example: Using the calculated pdf value between the vectorised point {0:2.0,1:10.0} and the center value (c) of the first centroid of the list {0:2.0,1:5.0}, now is calculated the final value of the pdf, using the formula:

  $$final\ pdf = (1 / (0.16666 + 0.83333)) * 0.16666 = 0.16666$$

  Afterwards, we do exactly the same calculations for the rest of the centroids, saving the results into a list, which finally is converted into a vector format:

  {0:0.16666,1:0.83333}

- After this step, it returns the vectorised format of its weight, which indicates the position – index of the maximum calculated probability (pdf) value.

> Example: From the previous example we can notice that the index-position with the greater value is the value with index one (1:), so the final weight of the data point {0:2.0,1:10.0} is {1:1.0}, which means that this point ({0:2.0,1:10.0}) from now on belongs to the cluster with identifier 0 (CL-0).

- Finally, it creates into the OLAP database the table "MAP", in which are stored in parallel from each thread, its assigned vectorised data points, with their IDs, the calculated maximum value index, which indicates in which cluster each data point belongs to, and their values. Thus, at this step, the Map phase prepared its output for the Reduce phase, which are the "MVALUES" and the "MKEYS values of the table "MAP", which describe the data points and the clusters that they belong to, respectively.

> Example: According to the table "POINTS" of the above example, we have five (5) records, which have been assigned into two (2) different threads, according to their IDs. As a result, in that case the first thread saves in the table "MAP" all its assigned data points with their maximum value indexes (IDs from 0-2) and the second one all the data points with IDs 3-4.

ID	MKEYS	MVALUES	COORDINATES
0	0	{0:2.0,1:10.0}	2.0, 10.0
3	0	{0:5.0,1:8.0}	5.0, 8.0
1	1	{0:2.0,1:5.0}	2.0, 5.0
4	1	{0:7.0,1:5.0}	7.0, 5.0
2	1	{0:8.0,1:4.0}	8.0, 4.0

❖ Execution of the Reduce Phase

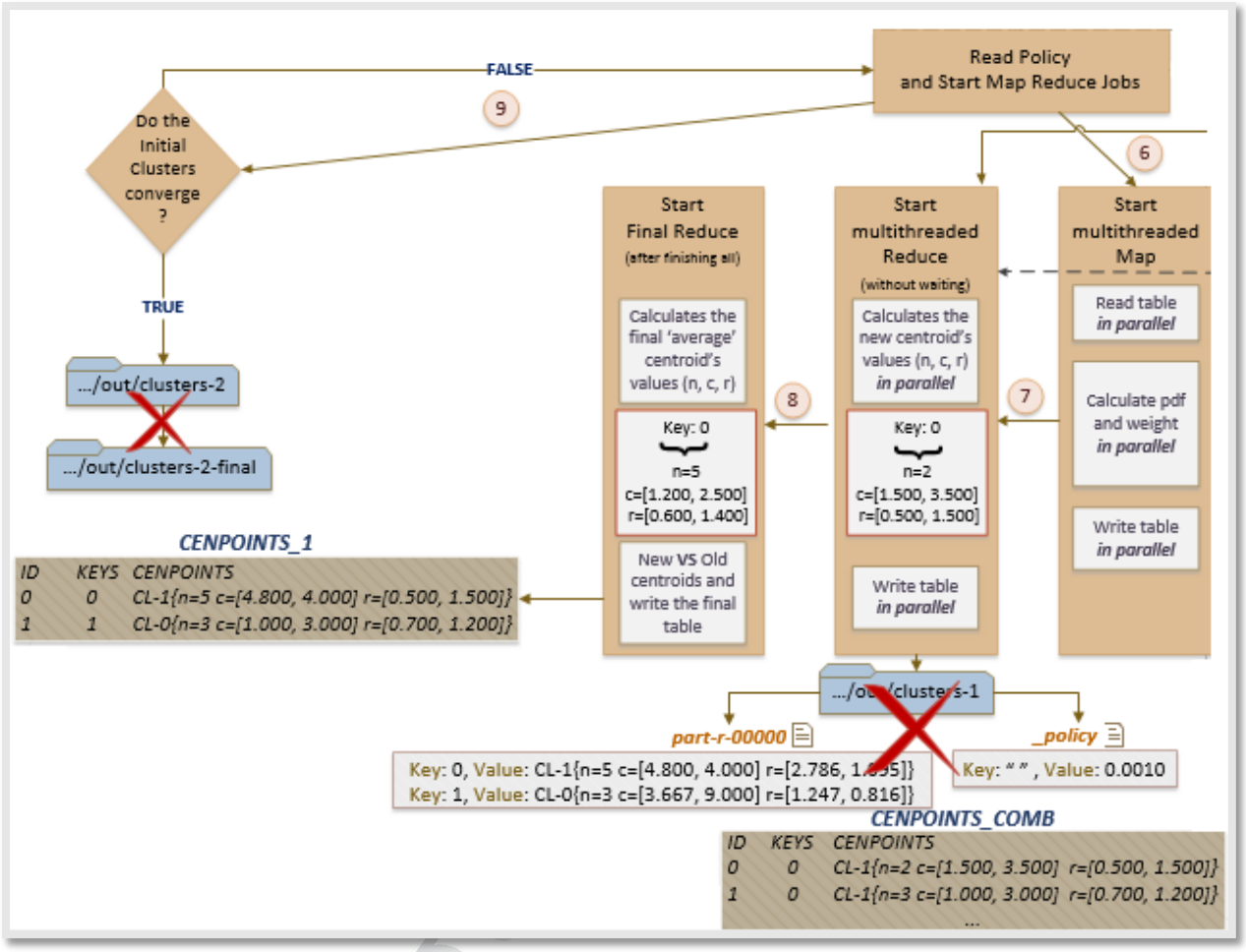


Figure 19 - Reduce Phase of K-Means using Mahout and an OLAP database

7 Having completed successfully the Map phase, the jobs of the Reduce phase take place. Like the Map phase, the Reduce does not use the MapReduce paradigm of Hadoop, but the new multithreaded implementation constructed by us, which uses exactly the same number of threads that were created during the Map phase. So, in the first step of the Reduce, we assign to each one of the existing threads to read the cluster information for each centroid contained in the table "CENPOINTS_0", save their cluster information into a list, and read the convergence delta value of the "POLICY" table.

Example: In the first iteration, each thread reads the values CL-1{n=0 c=[2.000, 5.000] r=[]} and CL-0{n=0 c=[2.000, 10.000] r=[]} of the table "CENPOINTS_0", saves them into its single list, and reads the value 0.0010 from the table "POLICY":

(for the first iteration)

ID	KEYS	CENPOINTS
0	CL-1	CL-1{n=0 c=[2.000, 5.000] r=[]}
1	CL-0	CL-0{n=0 c=[2.000, 10.000] r=[]}

(for the rest of the iterations)

ID	KEYS	CENPOINTS
0	0	CL-1{n=0 c=[2.000, 5.000] r=[]}
1	1	CL-0{n=0 c=[2.000, 10.000] r=[]}

[CL-1{n=0 c=[2.000, 5.000] r=[]}, CL-0{n=0 c=[2.000, 10.000] r=[]}]

ID	PAR	VALUE
0	ConvergenceDelta	0.0010

Then, each thread reads its corresponding data points of the "MAP" table and the current centroids of the iteration's "CENPOINTS_..." table. Taking one by one each cluster, each thread finds from its assigned data points of the table "MAP" all the data points that have the same MKEYS values, in order to combine them with each centroid that they read. More particularly, knowing the coordinates of the data points that belong to a cluster, and the cluster information about the current centroid, the new total number of observations (n), the new center (c) and the new radius (r) can be calculated for each centroid. The following example will make clear the whole procedure:

Example: Assume that the table "MAP" contains the following data points. As we can observe, the first thread reads the two points {0:2.0,1:5.0} and {0:8.0,1:4.0} that belong to the same cluster (cluster 0), as their MKEYS are the same (MKEYS = 1).

ID	MKEYS	MVALUES	COORDINATES
0	0	{0:2.0,1:10.0}	2.0, 10.0
3	0	{0:5.0,1:8.0}	5.0, 8.0
1	**1**	**{0:2.0,1:5.0}**	**2.0, 5.0**
4	1	{0:7.0,1:5.0}	7.0, 5.0
2	**1**	**{0:8.0,1:4.0}**	**8.0, 4.0**

Thus, in order to calculate the new values of the cluster 0 from the first thread, we have to do the following:

New total number of observations: **n** = 1 + 1 = 2
New center: **c** = [(2.0+8.0)/2, (5.0 + 4.0)/2] => **c** = [5.000, 4.500]
New radius: **r** = [sqrt.(((2.0-5.0)2 + (8.0-5.0)2)/2), sqrt.(((5.0-4.5)2 + (4.0-4.5)2)/2)]
=> **r** = [3.000, 0.707]

After the completion of the above steps, the new total number of observations (n), the new center (c) and the new radius (r) for each cluster from each thread are written in the OLAP database, in a new created table called "CENPOINTS_COMB".

Example: According to the two (2) created threads of the above example and the two (2) initial centroids, in the table "CENPOINTS_COMB" are written four (4) different values. In particular, each thread writes for each cluster its new information, according to the calculations that they have done in the previous step. Thus, the first thread writes the new cluster values about the two (2) clusters, and the second thread writes its own calculated new cluster values about the two (2) clusters.

It should be mentioned, that if one thread for a certain MKEY does not have any data points that belong to this exact cluster, then the thread keeps the cluster information of the initial centroid of the table "CENPOINTS_0", as happened with the CL-1 which was created from the second thread.

ID	KEYS	CENPOINTS
0	1	CL-1{n=2 c=[1.500, 3.500] r=[0.500, 1.500]}
1	0	CL-0{n=1 c=[2.000, 10.000] r=[]}
2	1	CL-1{n=0 c=[2.000, 5.000] r=[]}
3	0	CL-0{n=2 c=[4.500, 8.500] r=[0.500, 0.500]}

8 Afterwards the steps of the final Reduce phase take place, in which the calculated values of the new centroids by the several thread are going to merge. In this step a new table CENPOINTS_... is created, which refers to the iteration's final result. More specifically, in order to do this merge, the algorithm calculates for each pair of common centroids that exist in the table "CENPOINTS_COMB", the total number of observations (n), the average value of the center (c) and the average value of the radius (r).

Example: According to the cluster values of the cluster 0 CL-0{n=1 c=[2.000, 10.000] r=[]} and CL-0{n=2 c=[4.500, 8.500] r=[0.500, 0.500]} that were written in the table "CENPOINTS_COMB" from the two (2) existing threads, the new average values for the cluster 0 are calculated.

After this step, the algorithm uses the values of the current iteration center centroids (new centroids) and the previous iteration's center centroids (old centroids), in order to compare them with the value of the convergence delta. If the number that it calculated is <= of the convergence delta, then the centroids converge, and as a result, the new centroid takes as an identifier the value VL-… instead of CL-… . On the other hand, if the two compared centroids do not converge, then the new centroid continues having as an identifier the value CL-… . In particular the way that the convergence is calculated, is described in the following example:

Example: In the first iteration, cluster 0 (CL-0) had as an initial center the values [2.000, 10.000], while after the calculation of its new center which was made in the previous step, it had the value [2.000, 7.500]. Thus, in order to decide whether the cluster 0 has converged or not, we use the following formula:

$$converge = sqrt((2.0^2 + 7.5^2) - 2 * (2.0 * 2.0 + 10.0 * 7.5) + (2.0^2 + 10.0^2)) = 2.5$$

- if converge > convergence delta, then the new cluster does not converge with the old one, and the algorithm returns a "false" boolean value

- if converge <= convergence delta, then the new cluster converges with the old one, and the algorithm returns a "true" boolean value

Finally, having calculated all the new cluster information of the new centroids, by merging the different cluster values that each thread produced, we write the new centroids' cluster information into the iteration's table "CENPOINTS_...".

Example: In the first iteration, the table "CENPOINTS_1" is created, in which the new final centroids are stored:

ID	KEYS	CENPOINTS
0	0	CL-1{n=2 c=[4.500, 5.000] r=[2.500, 0.200]}
1	1	CL-0{n=2 c=[1.000, 8.000] r=[2.500, 1.000]}

Thus, at this moment the jobs of the Reduce phase are completed, giving as an output for the algorithm, the new calculated centroids.

9 After the successful creation of the table "CENPOINTS_...", a final basic step takes place, in which it is checked whether the new centroids have converged or not with the old ones. More specifically, this is only done by checking whether all the identifiers of the centroids have the value VL-... or not. As a consequence, in this step the algorithm checks whether the previously returned convergence boolean value has either the value "true" of "false". If all the centroids have as an identifier the value VL-... (value = "true"), then that means all the new centroids converge with the old ones, and the algorithm will not iterate again and it will rename the clusters-... folder to the folder clusters-...-final. Otherwise, the algorithm continues running, by iterating again from the step 6 and so on. However, it should me mentioned, that if at least one value has the identifier CL-... and all the other values have the identifier VL-..., then the algorithm will not stop, but it will continue running, until all the value clusters have converged (identifier VL-...).

❖ Completion of the algorithm

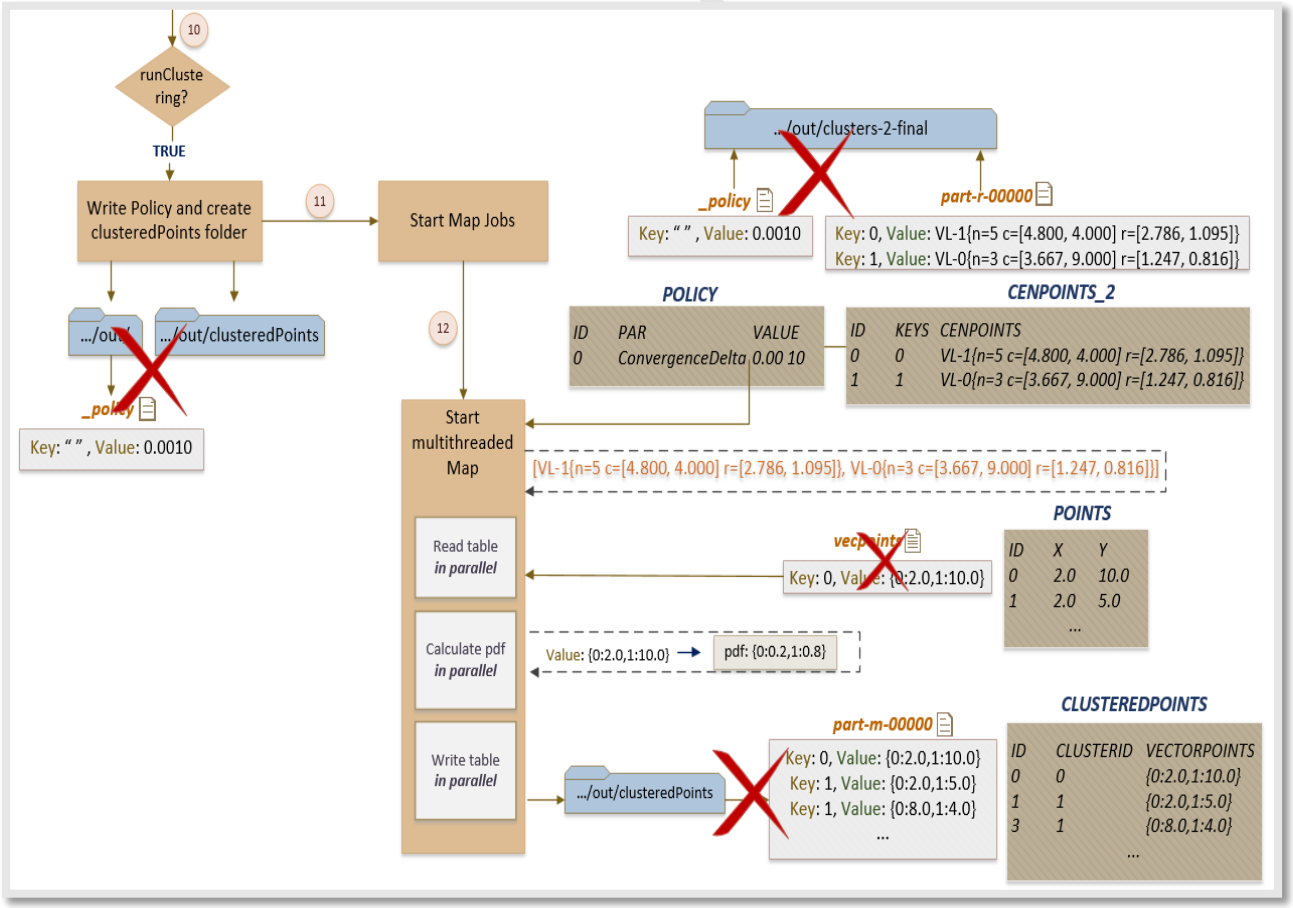


Figure 20 - Completion of K-Means using Mahout and an OLAP database

At this part of the program two different actions can be done, depending on the value of the input parameter runClustering of the algorithm:

- If *runClustering* equals with *true*, then the algorithm has to cluster the input data points after the iterations are completed, following a procedure that is going to be explained below.
- If *runClustering* equals with *false*, then the algorithm does not have to cluster the input data points after the iterations are completed, and the procedure of the clustering completes after the convergence check.

So, in the first case (runClustering = true) the procedure that the algorithm follows is:

12 In this step, the jobs of the final Map phase for the clustering of the data takes place. However, in this part of the program, it is not used the MapReduce paradigm of Hadoop, but another multithreaded implementation constructed by us. In particular, according to the number of the data points that are stored in the table "POINTS", there are created different threads to whom there is assigned a specific split part of the table, and they are intended to run in parallel, as it was done exactly during the Map phase. After the successful creation of the threads, each one of them reads the cluster information for each existing centroid from the final output table "CENPOINTS_..." of the Reduce phase, saves these information into a list, and reads the convergence delta value of the "POLICY" table.

Example: Each thread reads the CENPOINTS fields of the table "CENPOINTS_2", which contains the final converged centroids, saves them into its single list, and reads the value 0.0010 from the table "POLICY":

ID	KEYS	CENPOINTS
0	0	VL-1{n=3 c=[7.000, 4.333] r=[0.816, 0.471]}
1	1	VL-0{n=2 c=[1.500, 3.500] r=[0.500, 1.500]}

[VL-1{n=3 c=[7.000, 4.333] r=[0.816, 0.471]},VL-0{n=2 c=[1.500, 3.500] r=[0.500, 1.500]}]

ID	PAR	VALUE
0	ConvergenceDelta	0.0010

Then, the program assigns to each thread a specific part of the table "POINTS". More specifically, each thread is assigned to read in parallel the data point values between two (2) different IDs of the table "POINTS", using as a parameter a min and a max value.

Example: Assume again that the table "POINTS" has five (5) records, and that each thread is assigned to read only three (3) records. Thus, as we have five (5) records, two (2) different threads are going to be created, each one of them reading three (3) different records, according to their IDs. As a result, the first thread reads all the records with IDs 0-2 and the second one the records with the IDs 3-4.

ID	COLUMN_0	COLUMN_1
0	2.0	10.0
1	2.0	5.0
2	8.0	4.0
3	5.0	8.0
4	7.0	5.0

Thus, for each data point that each thread reads, it implements the following procedure:

- It calculates the pdf (probability density function -probability of a vectorised point to belong to a cluster-) from each centroid that exists in the above list, and saves all the calculated probabilities into a list. In order to calculate the pdf, it uses an exact formula, which will be explained using the following example.

  Example: During the first iteration over the input file, the algorithm reads the vectorised point {0:2.0,1:10.0}. Comparing this with the center value (c) of the first centroid of the list {0:2.0,1:5.0}, is calculated the pdf between this given point and the centroid, using the formula:

  $$pdf = 1 / (1 + sqrt((2.0-2.0)^2 + (10.0-5.0)^2)) = 0.16666$$

  Afterwards, we do exactly the same calculations for the rest of the centroids, saving the results into a list:

  [0.16666, 0.83333]

- Then, it converts the list with the calculated pdfs into a vector format, and using this vector, it calculates the final pdf value of each cluster, it adds it to a list, and finally converts this list with the final pdfs into a vector format. The formula that is used in order to calculate the final value of the pdf will be explained below with an example:

> Example: Using the calculated pdf value between the vectorised point {0:2.0,1:10.0} and the center value (c) of the first centroid of the list {0:2.0,1:5.0}, now is calculated the final value of the pdf, using the formula:
>
> final pdf = (1 / (0.16666 + 0.83333)) * 0.16666 = 0.16666
>
> Afterwards, we do exactly the same calculations for the rest of the centroids, saving the results into a list, which finally is converted into a vector format:
>
> {0:0.16666,1:0.83333}

- Finally, it creates into the table "CLUSTEREDPOINTS", in which are stored in parallel from each thread their assigned vectorised data points with theirs IDs, accompanied with their calculated cluster key.

> Example: According to table "CLUSTEREDPOINTS" of the above example, we have five (5) records, which have been assigned into two (2) different threads, according to their IDs. As a result, in that case, the first thread saves in the table "CLUSTEREDPOINTS" all its assigned data points with their maximum value indexes (IDs from 0-2) and the second one all the data points with IDs 3-4.

ID	CLUSTERID	VECTORPOINTS
0	0	{0:2.0,1:10.0}
3	0	{0:5.0,1:8.0}
1	1	{0:2.0,1:5.0}
4	1	{0:7.0,1:5.0}
2	1	{0:8.0,1:4.0}

Thus, the final results of all this procedure are the "CLUSTEREDPOINTS" output table of the single Map phase, which contains all the input data points grouped into clusters and the "CENPOINTS_..." output table of the Reduce phase, which contains all the final clusters. For that reason, the final result that is provided to the user answers to the question of which point belongs to which cluster and which are the final created clusters.

Chapter 6

Evaluation

In this chapter, take place all the experiments that were made in order to verify that both of the implementations work properly and successfully produce their final results. More particularly, in the beginning of the chapter are described the characteristics of the hardware and software that were used during the execution of the experiments, and after this, the individual experiments that made are stated. Finally, we evaluate the results and the performance of the two implementations, by comparing their results.

6.1. Experimental Environment

This section describes the hardware-software apparatus used to develop the two different implementations of K-Means algorithm, for clustering data. In particular, all the implementations and the experiments that were executed, ran in a single computer with the following specifications (Table 5 and Table 6):

	Specifications	Values
Hardware	Operating System	Ubuntu 10.04 (lucid)
	Memory	11.8 GiB
	Processor	Intel® Core ™ i7 CPU 920 @ 2.67 GHz x 8
	OS type	64-bit
	Disk	2169.1 GiB

Table 5 - Hardware Specifications

	Mahout with HDFS	Mahout with DERBY
Software	Mahout environment version 0.9	Mahout environment version 0.9
	Hadoop environment version 1.2.1	DERBY DBMS version 10.11.1.1
	Single-node Hadoop installation	

Table 6 - Software Specifications

At this points it should be mentioned that for the experiments we used for storage the DERBY centralized database instead of the OLAP distributed database, as our first goal was to check whether the new implementation was working properly or not and whether its final clustering results were correct or not, running it with small easy-to-explain datasets.

Moreover, in order to be developed both implementations and run their experiments, the installation of Java (version 7) was required, accompanied with the installation of Maven (version 3). Finally, it should be mentioned that both implementations were written in Java, using the Netbeans IDE (version 7.2).

6.2. Dataset Description

In order to evaluate the two implementations that were developed, some experiments should take place. In both implementations we run experiments using a common dataset, with two different sizes. More particularly, the one dataset was relatively smaller than the other one (citizens_1000) having as a size approximately 30KB, whilst the other dataset was about 6MB (citizens_20000).

The dataset that was used is a randomly created dataset by a java program made by us, which contains information about the habits of a town's citizens, considering them as smokers and alcoholics, or non-smokers and non-alcoholics, correspondingly. Furthermore, the smaller dataset contains information for 1000 citizens of the town, whilst the bigger one contains information for 20000 of them. In both of them, each record consists of 11 different attributes each one of them having its own different range of values. This means in other words, that the dataset consists of 11 different dimensions.

Finally, it should be mentioned that all the containing values of the dataset are only numerical non-noisy values, where each attribute is separated from the others with commas (",").

To sum up, in Table 7 are described all the containing attributes of the dataset, with their corresponding names and ranges of values:

	Dataset of a town's population for smokers and alcoholics										
Description	ID	Gender	Age	Weight (kg)	Height (cm)	Married	Children	Educational Level	Work	Smoke	Drink
Values	1…	1-2	18-70	1-150	60-200	1-2	1-5	1-7	1-2	1-2	1-4

Table 7 - Dataset Attributes

More specifically, the aforementioned attributes represent the following:

- *ID:* Starts from 1 and is auto increment.
- *Gender:* Holds values either 1 (male) or 2 (female).
- *Age:* Holds values between 18 and 70.
- *Weight:* Holds values between 1 and 150 (in kg).
- *Height:* Holds values between 60 and 200 (in cm).
- *Married:* Holds values either 1 (no) or 2 (yes).
- *Children:* Holds values 1 (zero) or 2 (one) or 3 (two) or 4 (three) or 5 (4 or more).
- *Educational Level:* Holds values 1 (no education) or 2 (primary) or 3 (high school) or 4 (bachelor) or 5 (master) or 6 (phd) or 7 (higher than the above).
- *Work:* Holds values either 1 (no) or 2 (yes).
- *Smoke:* Holds values either 1 (no) or 2 (yes).
- *Drink:* Holds values 1 (never) or 2 (1 time/week) or 3 (3-4 times/week) or 4 (everyday).

A small sample of the dataset can be seen in Figure 21, where someone can observe the 11 different attributes of each line, with their values ranging between the values that were mentioned above.

```
1,1,33,72,109,1,4,1,1,1,3
2,2,66,4,79,1,2,4,2,2,1
3,1,65,12,80,2,4,7,2,2,2
4,2,26,92,52,2,4,3,2,2,4
5,1,68,43,58,2,3,2,1,2,2
6,2,57,147,55,1,4,1,2,2,4
7,1,63,105,108,1,4,3,1,1,2
8,2,42,115,127,2,1,6,2,2,3
9,1,34,83,35,2,2,5,2,1,2
10,2,47,18,104,1,3,5,2,2,3
11,2,26,50,129,1,1,2,1,1,2
12,2,29,99,5,2,5,2,2,1,3
13,1,53,105,82,1,1,1,2,1,1
14,1,22,125,21,2,1,5,2,1,3
15,1,64,55,75,2,1,7,1,1,2
```

Figure 21 - Sample of the dataset

6.3. Experiments of the K-Means Implementations

Having created the required dataset for testing the performance of the two implementations of the K-Means algorithm using Mahout's libraries both on top of HDFS and on top of DERBY, the description of the two experiments that took place, is going to be stated in this section. In particular, in each experiment that was made, is given the algorithm's input parameters that were used and the final output results of the execution of the algorithm, accompanied with its total execution time.

❖ **First Dataset's Experiment**

As mentioned above, the dataset that was used for the first experiment, contained 1000 different data records, where the values of each line were randomly created, according to the attributes that were described earlier. Thus, using this dataset, we run the experiments both on top of DERBY and on top of HDFS, customizing the algorithms to run with the same input parameters. As a result of this, in the final phase of the experiments, we will be able to make a comparison between the two implementations according to their results.

- **K-Means algorithm on top of HDFS**

According to the parameters that the K-Means algorithm needs as an input when it is implemented on top of HDFS (section 5.2.), in that scenario were used the ones stated below. It should be mentioned that the size of the citizens_1000 input dataset was about 30KB, thus the file stored in HDFS only partitioned into one block.

number of clusters ***k = 5***

• *input:* …/in/citizens_1000	• *maxIterations:* 20
• *clustersIn:* …/in/part-00000	• *runClustering:* true
• *output:* …/out/	• *clusterClassificationThreshold:* 0
• *convergenceDelta:* 0.0010	• *runSequential:* true

As a result of all these, after running the experiment with these parameters, if we browse the HDFS file system, we can see that the input and output folders have been successfully created, accompanied with the input file that was uploaded to the HDFS, partitioned indeed into one block (Figure 22).

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
in	dir				2015-02-06 18:43	rwxr-xr-x	iro	supergroup
out	dir				2015-02-06 18:43	rwxr-xr-x	iro	supergroup
citizens_1000	file	32.01 KB	1	64 MB	2015-02-06 18:43	rw-r--r--	iro	supergroup

Figure 22 - Uploaded dataset to HDFS

Opening the input folder "in", we can observe that all the mandatory input files have been created (Figure 23).

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
part-00000	file	2.68 KB	1	64 MB	2015-02-06 18:43	rw-r--r--	iro	supergroup
vecpoints	file	120.37 KB	1	64 MB	2015-02-06 18:43	rw-r--r--	iro	supergroup

Figure 23 - Input created folder to HDFS

By opening the output folder "out", we can see that several clusters folder for each iteration have been created (Figure 24).

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_policy	file	0.19 KB	1	64 MB	2015-02-06 18:48	rw-r--r--	iro	supergroup
clusteredPoints	dir				2015-02-06 18:48	rwxr-xr-x	iro	supergroup
clusters-0	dir				2015-02-06 18:43	rwxr-xr-x	iro	supergroup
clusters-1	dir				2015-02-06 18:43	rwxr-xr-x	iro	supergroup
clusters-10	dir				2015-02-06 18:45	rwxr-xr-x	iro	supergroup
clusters-11	dir				2015-02-06 18:46	rwxr-xr-x	iro	supergroup
clusters-12	dir				2015-02-06 18:46	rwxr-xr-x	iro	supergroup
clusters-13	dir				2015-02-06 18:46	rwxr-xr-x	iro	supergroup
clusters-14	dir				2015-02-06 18:47	rwxr-xr-x	iro	supergroup
clusters-15	dir				2015-02-06 18:47	rwxr-xr-x	iro	supergroup
clusters-16	dir				2015-02-06 18:47	rwxr-xr-x	iro	supergroup
clusters-17	dir				2015-02-06 18:47	rwxr-xr-x	iro	supergroup
clusters-18	dir				2015-02-06 18:48	rwxr-xr-x	iro	supergroup
clusters-19	dir				2015-02-06 18:48	rwxr-xr-x	iro	supergroup
clusters-2	dir				2015-02-06 18:43	rwxr-xr-x	iro	supergroup
clusters-20-final	dir				2015-02-06 18:48	rwxr-xr-x	iro	supergroup
clusters-3	dir				2015-02-06 18:43	rwxr-xr-x	iro	supergroup
clusters-4	dir				2015-02-06 18:44	rwxr-xr-x	iro	supergroup
clusters-5	dir				2015-02-06 18:44	rwxr-xr-x	iro	supergroup
clusters-6	dir				2015-02-06 18:44	rwxr-xr-x	iro	supergroup
clusters-7	dir				2015-02-06 18:45	rwxr-xr-x	iro	supergroup
clusters-8	dir				2015-02-06 18:45	rwxr-xr-x	iro	supergroup
clusters-9	dir				2015-02-06 18:45	rwxr-xr-x	iro	supergroup

Figure 24 - Output created folder to HDFS

If we open each one of these folders, we can see that they contain the required output sequence files (Figure 25).

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 KB	1	64 MB	2015-02-06 18:43	rw-r--r--	iro	supergroup
_logs	dir				2015-02-06 18:43	rwxr-xr-x	iro	supergroup
_policy	file	0.2 KB	1	64 MB	2015-02-06 18:43	rw-r--r--	iro	supergroup
part-r-00000	file	2.96 KB	1	64 MB	2015-02-06 18:43	rw-r--r--	iro	supergroup

Figure 25 - Files of output folder to HDFS

Finally, the final output result that appeared to the user after the execution of the current experiment, was the following:

```
---Final Clusters---

CL-0{n=224 c=[1.509, 46.281, 115.683, 112.138, 1.411, 2.987, 3.911, 1.496, 1.527,
 2.504] r=[0.500, 14.596, 20.094, 18.677, 0.492, 1.450, 1.980, 0.500, 0.499, 1.11
8]}
CL-1{n=170 c=[1.576, 45.300, 26.024, 45.259, 1.512, 2.882, 3.988, 1.512, 1.512, 2
.400] r=[0.494, 15.691, 14.980, 22.978, 0.500, 1.397, 2.041, 0.500, 0.500, 1.103]
}
CL-2{n=236 c=[1.504, 42.419, 42.292, 109.169, 1.551, 2.911, 4.047, 1.492, 1.542,
2.576] r=[0.500, 14.289, 21.910, 17.313, 0.497, 1.352, 2.094, 0.500, 0.498, 1.089
]}
CL-3{n=167 c=[1.437, 42.587, 127.413, 39.898, 1.485, 2.946, 3.934, 1.473, 1.425,
2.431] r=[0.496, 15.642, 14.388, 22.260, 0.500, 1.372, 2.012, 0.499, 0.494, 1.097
]}
CL-4{n=203 c=[1.493, 43.094, 77.222, 33.655, 1.478, 3.133, 3.956, 1.502, 1.483, 2
.557] r=[0.500, 15.348, 14.825, 21.230, 0.500, 1.389, 1.986, 0.500, 0.500, 1.123]
}

Total number of instances: 1000
Total time of execution: 353 sec
```

Total number of iterations: **20**
Total time of execution: **353 sec**
Converged: **false**

Table 8 - K-Means results on top of HDFS for k=5 and instances=1000

- **K-Means algorithm on top of DERBY**

According to the parameters that the K-Means algorithm needs as an input when is implementing on top of a database (section 5.3.), in that scenario we used the following ones:

number of clusters ***k = 5***	
• *connection:* …/derby/mahdb	• *clusterClassificationThreshold:* 0
• *convergenceDelta:* 0.0010	• *numberofpoints:* 1000
• *maxIterations:* 20	• *k:* 5
• *runClustering:* true	• *points/thread*: 100

The final output result that appeared to the user after the execution of the current experiment, was the following:

```
---Final Clusters---

CL-0{n=224 c=[1.509, 46.281, 115.683, 112.138, 1.411, 2.987, 3.911, 1.496, 1.527,
 2.504] r=[0.500, 14.596, 20.094, 18.677, 0.492, 1.450, 1.980, 0.500, 0.499, 1.11
3]}
CL-1{n=170 c=[1.576, 45.300, 26.024, 45.259, 1.512, 2.882, 3.988, 1.512, 1.512, 2
.400] r=[0.494, 15.691, 14.980, 22.978, 0.500, 1.397, 2.041, 0.500, 0.500, 1.103]
}
CL-2{n=236 c=[1.504, 42.419, 42.292, 109.169, 1.551, 2.911, 4.047, 1.492, 1.542,
2.576] r=[0.500, 14.289, 21.910, 17.313, 0.497, 1.352, 2.094, 0.500, 0.498, 1.089
]}
CL-3{n=167 c=[1.437, 42.587, 127.413, 39.898, 1.485, 2.946, 3.934, 1.473, 1.425,
2.431] r=[0.496, 15.642, 14.388, 22.260, 0.500, 1.372, 2.012, 0.499, 0.494, 1.097
]}
CL-4{n=203 c=[1.493, 43.094, 77.222, 33.655, 1.478, 3.133, 3.956, 1.502, 1.483, 2
.557] r=[0.500, 15.348, 14.825, 21.230, 0.500, 1.389, 1.986, 0.500, 0.500, 1.123]
}

Total number of instances: 1000
Total time of execution: 59 sec
```

Total number of iterations: **20**
Total time of execution: **59 sec**
Converged: **false**

Table 9 - K-Means results on top of DERBY for k=5 and instances=1000

- **K-Means algorithm on top of HDFS**

According to the parameters that the K-Means algorithm needs as an input when is implementing on top of HDFS (section 5.2.), in that scenario we used the following ones:

number of clusters ***k = 20***	
• *input:* …/in/citizens_1000	• *maxIterations:* 20
• *clustersIn:* …/in/part-00000	• *runClustering:* true
• *output:* …/out/	• *clusterClassificationThreshold:* 0
• *convergenceDelta:* 0.0010	• *runSequential:* true

The final output result that appeared after the execution of the current experiment was:

```
CL-14{n=52 c=[1.462, 53.019, 66.308, 40.596, 1.538, 3.269, 4.231, 1.423, 1.385, 2
.712] r=[0.499, 11.615, 10.566, 9.865, 0.499, 1.416, 1.977, 0.494, 0.487, 1.006]}
VL-15{n=39 c=[1.513, 28.923, 25.718, 41.949, 1.692, 3.179, 3.667, 1.359, 1.564, 2
.282] r=[0.500, 8.260, 14.071, 11.507, 0.462, 1.465, 1.899, 0.480, 0.496, 0.986]}
VL-16{n=43 c=[1.372, 58.977, 129.767, 22.674, 1.488, 2.674, 4.465, 1.512, 1.442,
2.302] r=[0.483, 7.366, 12.338, 13.494, 0.500, 1.307, 1.834, 0.500, 0.497, 1.090]
}
CL-17{n=37 c=[1.459, 33.432, 103.622, 115.919, 1.459, 3.108, 4.000, 1.405, 1.541,
 2.595] r=[0.498, 8.446, 10.916, 11.835, 0.498, 1.410, 1.959, 0.491, 0.498, 1.026
]}
CL-18{n=41 c=[1.610, 57.829, 61.024, 119.024, 1.439, 3.000, 4.049, 1.488, 1.463,
2.512] r=[0.488, 7.378, 12.826, 11.230, 0.496, 1.448, 2.163, 0.500, 0.499, 1.085]
}
VL-19{n=44 c=[1.455, 36.205, 35.023, 116.364, 1.568, 2.909, 3.886, 1.500, 1.545,
2.591] r=[0.498, 10.439, 7.325, 13.481, 0.495, 1.221, 1.933, 0.500, 0.498, 1.073]
}

Total number of instances: 1000
Total time of execution: 353 sec
```

Total number of iterations: **20**
Total time of execution: **353 sec**
Converged: **false**

Table 10 - K-Means results on top of HDFS for k=20 and instances=1000

- **K-Means algorithm on top of DERBY**

According to the parameters that the K-Means algorithm needs as an input when is implementing on top of a database (section 5.3.), in that scenario we used the following ones:

number of clusters ***k = 20***	
• *connection:* …/derby/mahdb	• *clusterClassificationThreshold:* 0
• *convergenceDelta:* 0.0010	• *numberofpoints:* 1000
• *maxIterations:* 20	• *k:* 20
• *runClustering:* true	• *points/thread*: 100

The final output result that appeared to the user after the execution of the current experiment, was the following:

```
CL-14{n=52 c=[1.462, 53.019, 66.308, 40.596, 1.538, 3.269, 4.231, 1.423, 1.385, 2
.712] r=[0.499, 11.615, 10.566, 9.865, 0.499, 1.416, 1.977, 0.494, 0.487, 1.006]}
VL-15{n=39 c=[1.513, 28.923, 25.718, 41.949, 1.692, 3.179, 3.667, 1.359, 1.564, 2
.282] r=[0.500, 8.260, 14.071, 11.507, 0.462, 1.465, 1.899, 0.480, 0.496, 0.986]}
VL-16{n=43 c=[1.372, 58.977, 129.767, 22.674, 1.488, 2.674, 4.465, 1.512, 1.442,
2.302] r=[0.483, 7.366, 12.338, 13.494, 0.500, 1.307, 1.834, 0.500, 0.497, 1.090]
}
CL-17{n=37 c=[1.459, 33.432, 103.622, 115.919, 1.459, 3.108, 4.000, 1.405, 1.541,
 2.595] r=[0.498, 8.446, 10.916, 11.835, 0.498, 1.410, 1.959, 0.491, 0.498, 1.026
]}
CL-18{n=41 c=[1.610, 57.829, 61.024, 119.024, 1.439, 3.000, 4.049, 1.488, 1.463,
2.512] r=[0.488, 7.378, 12.826, 11.230, 0.496, 1.448, 2.163, 0.500, 0.499, 1.085]
}
VL-19{n=44 c=[1.455, 36.205, 35.023, 116.364, 1.568, 2.909, 3.886, 1.500, 1.545,
2.591] r=[0.498, 10.439, 7.325, 13.481, 0.495, 1.221, 1.933, 0.500, 0.498, 1.073]
}

Total number of instances: 1000
Total time of execution: 59 sec
```

Total number of iterations: **20**
Total time of execution: **59 sec**
Converged: **false**

Table 11 - K-Means results on top of DERBY for k=20 and instances=1000

❖ **Second Dataset's Experiment**

The dataset that was used for the second experiment, contained 20000 different data records, where the values of each line were randomly created, according to the attributes that were described earlier. Thus, in this experiment we followed exactly the same procedure as we did with the first experiment.

- **K-Means algorithm on top of HDFS**

According to the parameters that the K-Means algorithm needs as an input when is implementing on top of HDFS (section 5.2.), in that scenario we used the following ones:

number of clusters ***k = 5***

• *input:* …/in/citizens_20000	• *maxIterations:* 20
• *clustersIn:* …/in/part-00000	• *runClustering:* true
• *output:* …/out/	• *clusterClassificationThreshold:* 0
• *convergenceDelta:* 0.0010	• *runSequential:* true

The final output result that appeared after the execution of the current experiment was:

```
---Final Clusters---

CL-0{n=4777 c=[1.498, 44.318, 112.632, 108.589, 1.487, 2.978, 4.041, 1.502, 1.488
, 2.508] r=[0.500, 15.300, 21.494, 19.131, 0.500, 1.407, 1.985, 0.500, 0.500, 1.1
15]}
CL-1{n=3687 c=[1.500, 43.462, 74.756, 40.637, 1.505, 3.001, 3.976, 1.505, 1.497,
2.495] r=[0.500, 15.264, 14.108, 22.148, 0.500, 1.398, 2.015, 0.500, 0.500, 1.119
]}
CL-2{n=3453 c=[1.494, 44.197, 24.563, 36.048, 1.506, 3.040, 4.007, 1.492, 1.512,
2.504] r=[0.500, 15.249, 14.307, 21.146, 0.500, 1.413, 2.021, 0.500, 0.500, 1.114
]}
CL-3{n=4554 c=[1.509, 43.881, 36.704, 108.642, 1.494, 3.006, 3.974, 1.494, 1.484,
 2.507] r=[0.500, 15.193, 20.740, 19.364, 0.500, 1.398, 2.009, 0.500, 0.500, 1.11
6]}
CL-4{n=3529 c=[1.503, 43.818, 125.766, 37.876, 1.498, 2.987, 3.977, 1.495, 1.496,
 2.496] r=[0.500, 15.258, 14.374, 21.239, 0.500, 1.427, 2.003, 0.500, 0.500, 1.11
3]}

Total number of instances: 20000
Total time of execution: 399 sec
```

Total number of iterations: **20**
Total time of execution: **399 sec**
Converged: **false**

Table 12 - K-Means results on top of HDFS for k=5 and instances=20000

- **K-Means algorithm on top of DERBY**

According to the parameters that the K-Means algorithm needs as an input when is implementing on top of a database (section 5.3.), in that scenario we used the following ones:

number of clusters ***k = 5***	
• *connection:* …/derby/mahdb	• *clusterClassificationThreshold:* 0
• *convergenceDelta:* 0.0010	• *numberofpoints:* 20000
• *maxIterations:* 20	• *k:* 5
• *runClustering:* true	• *points/thread*: 1000

The final output result that appeared to the user after the execution of the current experiment, was the following:

```
---Final Clusters---

CL-0{n=4777 c=[1.498, 44.318, 112.632, 108.589, 1.487, 2.978, 4.041, 1.502, 1.488
, 2.508] r=[0.500, 15.300, 21.494, 19.131, 0.500, 1.407, 1.985, 0.500, 0.500, 1.1
15]}
CL-1{n=3687 c=[1.500, 43.462, 74.756, 40.637, 1.505, 3.001, 3.976, 1.505, 1.497,
2.495] r=[0.500, 15.264, 14.108, 22.148, 0.500, 1.398, 2.015, 0.500, 0.500, 1.119
]}
CL-2{n=3453 c=[1.494, 44.197, 24.563, 36.048, 1.506, 3.040, 4.007, 1.492, 1.512,
2.504] r=[0.500, 15.249, 14.307, 21.146, 0.500, 1.413, 2.021, 0.500, 0.500, 1.114
]}
CL-3{n=4554 c=[1.509, 43.881, 36.704, 108.642, 1.494, 3.006, 3.974, 1.494, 1.484,
 2.507] r=[0.500, 15.193, 20.740, 19.364, 0.500, 1.398, 2.009, 0.500, 0.500, 1.11
6]}
CL-4{n=3529 c=[1.503, 43.818, 125.766, 37.876, 1.498, 2.987, 3.977, 1.495, 1.496,
 2.496] r=[0.500, 15.258, 14.374, 21.239, 0.500, 1.427, 2.003, 0.500, 0.500, 1.11
3]}

Total number of instances: 20000
Total time of execution: 108 sec
```

Total number of iterations: **20**
Total time of execution: **108 sec**
Converged: **false**

Table 13 - K-Means results on top of DERBY for k=5 and instances=20000

- **K-Means algorithm on top of HDFS**

According to the parameters that the K-Means algorithm needs as an input when is implementing on top of HDFS (section 5.2.), in that scenario we used the following ones:

number of clusters ***k = 20***	
• *input:* …/in/citizens_20000	• *maxIterations:* 20
• *clustersIn:* …/in/part-00000	• *runClustering:* true
• *output:* …/out/	• *clusterClassificationThreshold:* 0
• *convergenceDelta:* 0.0010	• *runSequential:* true

The final output result that appeared after the execution of the current experiment was:

```
CL-14{n=1177 c=[1.496, 38.893, 91.218, 17.624, 1.509, 2.951, 3.948, 1.496, 1.500,
 2.505] r=[0.500, 13.817, 11.378, 10.565, 0.500, 1.414, 2.024, 0.500, 0.500, 1.09
6]}
CL-15{n=1031 c=[1.500, 54.876, 99.420, 51.938, 1.504, 3.031, 3.952, 1.519, 1.503,
 2.532] r=[0.500, 10.783, 11.888, 11.647, 0.500, 1.392, 2.025, 0.500, 0.500, 1.11
1]}
CL-16{n=996 c=[1.491, 38.974, 92.488, 125.727, 1.492, 2.939, 3.988, 1.509, 1.515,
 2.546] r=[0.500, 12.667, 10.794, 10.117, 0.500, 1.401, 1.965, 0.500, 0.500, 1.10
4]}
CL-17{n=888 c=[1.479, 30.367, 94.356, 80.868, 1.499, 3.046, 3.959, 1.514, 1.481,
2.521] r=[0.500, 8.203, 12.250, 11.682, 0.500, 1.407, 1.956, 0.500, 0.500, 1.157]
}
CL-18{n=1056 c=[1.489, 55.975, 58.566, 24.993, 1.500, 3.040, 4.054, 1.482, 1.480,
 2.490] r=[0.500, 9.339, 11.535, 12.780, 0.500, 1.424, 2.037, 0.500, 0.500, 1.122
]}
CL-19{n=870 c=[1.516, 57.747, 94.086, 98.100, 1.476, 3.010, 4.057, 1.506, 1.485,
2.528] r=[0.500, 8.185, 11.993, 11.102, 0.499, 1.397, 1.994, 0.500, 0.500, 1.137]
}

Total number of instances: 20000
Total time of execution: 429 sec
```

Total number of iterations: **20**
Total time of execution: **429 sec**
Converged: **false**

Table 14 - K-Means results on top of HDFS for k=20 and instances=20000

- **K-Means algorithm on top of DERBY**

According to the parameters that the K-Means algorithm needs as an input when is implementing on top of a database (section 5.3.), in that scenario we used the following ones:

number of clusters ***k = 20***	
• *connection:* …/derby/mahdb	• *clusterClassificationThreshold:* 0
• *convergenceDelta:* 0.0010	• *numberofpoints:* 20000
• *maxIterations:* 20	• *k:* 20
• *runClustering:* true	• *points/thread*: 1000

The final output result that appeared to the user after the execution of the current experiment, was the following:

```
CL-14{n=1177 c=[1.496, 38.893, 91.218, 17.624, 1.509, 2.951, 3.948, 1.496, 1.500,
 2.505] r=[0.500, 13.817, 11.378, 10.565, 0.500, 1.414, 2.024, 0.500, 0.500, 1.09
6]}
CL-15{n=1031 c=[1.500, 54.876, 99.420, 51.938, 1.504, 3.031, 3.952, 1.519, 1.503,
 2.532] r=[0.500, 10.783, 11.888, 11.647, 0.500, 1.392, 2.025, 0.500, 0.500, 1.11
1]}
CL-16{n=996 c=[1.491, 38.974, 92.488, 125.727, 1.492, 2.939, 3.988, 1.509, 1.515,
 2.546] r=[0.500, 12.667, 10.794, 10.117, 0.500, 1.401, 1.965, 0.500, 0.500, 1.10
4]}
CL-17{n=888 c=[1.479, 30.367, 94.356, 80.868, 1.499, 3.046, 3.959, 1.514, 1.481,
2.521] r=[0.500, 8.203, 12.250, 11.682, 0.500, 1.407, 1.956, 0.500, 0.500, 1.157]
}
CL-18{n=1056 c=[1.489, 55.975, 58.566, 24.993, 1.500, 3.040, 4.054, 1.482, 1.480,
 2.490] r=[0.500, 9.339, 11.535, 12.780, 0.500, 1.424, 2.037, 0.500, 0.500, 1.122
]}
CL-19{n=870 c=[1.516, 57.747, 94.086, 98.100, 1.476, 3.010, 4.057, 1.506, 1.485,
2.528] r=[0.500, 8.185, 11.993, 11.102, 0.499, 1.397, 1.994, 0.500, 0.500, 1.137]
}

Total number of instances: 20000
Total time of execution: 692 sec
```

Total number of iterations: **20**
Total time of execution: **692 sec**
Converged: **false**

Table 15 - K-Means results on top of DERBY for k=20 and instances=20000

6.4. Evaluation – Comparison of the Results

After the execution of all the aforementioned experiments, the evaluation and the comparison of their results take place. In this stage we evaluate and interpret the final output results, and we state the main differences between the two implementations, based upon their final results and performances.

- **First Dataset's Experiment**

During the first experiment, the same dataset (*1000 numerical non-noisy records*) used for both implementations of K-Means. However, even though the same dataset was used for the experiments, as someone could imagine, the performance between the two implementations was quite different, but always concluding in exactly the same final results (final clusters). In particular, the characteristics of the performance of each implementation, can be viewed in Table 16:

K-Means on top of HDFS	**K-Means on top of DERBY**
number of clusters ***k = 5***	
Total time of execution: **353 sec**	Total time of execution: **59 sec**
Total iterations: **20**	Total iterations: **20**
Final Clusters Converged: **False**	Final Clusters Converged: **False**
number of clusters ***k = 20***	
Total time of execution: **353 sec**	Total time of execution: **59 sec**
Total iterations: **20**	Total iterations: **20**
Final Clusters Converged: **False**	Final Clusters Converged: **False**

Table 16 - Implementations' comparison for instances = 1000

Observing these results, someone could see that the K-Means implementation on top of DERBY is 5 times faster than the other implementation, regardless of the chosen number of clusters, as in both scenarios (k=5 and k=20) the total execution time is exactly the same. Moreover, as mentioned above, the output final clusters have exactly the same values, without having converged clusters until the maximum number of the twenty (20) iterations.

❖ **Second Dataset's Experiment**

During the second experiment, the same dataset (*20000 numerical non-noisy records*) used for both implementations of the K-Means algorithm. Despite the fact that the same dataset was used for the experiments, the performance between the two implementations was quite different, but always concluding in exactly the same final results (final clusters), as happened during the first experiment. More specifically, the characteristics of the performance of each implementation, can be viewed in Table 17:

K-Means on top of HDFS	**K-Means on top of DERBY**
number of clusters ***k = 5***	
Total time of execution: **399 sec**	Total time of execution: **108 sec**
Total iterations: **20**	Total iterations: **20**
Final Clusters Converged: **False**	Final Clusters Converged: **False**
number of clusters ***k = 20***	
Total time of execution: **429 sec**	Total time of execution: **692 sec**
Total iterations: **20**	Total iterations: **20**
Final Clusters Converged: **False**	Final Clusters Converged: **False**

Table 17 - Implementations' comparison for instances = 20000

Observing the results, someone could see that the K-Means implementation on top of DERBY is 3 times faster than the other implementation, when the number of clusters equals with 5 (k=5). On the other hand, when the number of clusters equals with 20 (k=20), we can observe that the HDFS implementation is 1.5 times faster than this of the DERBY. However, irrespectively of the total execution time of each implementation, the output final clusters had exactly the same values, without having converged clusters until the maximum number of the twenty (20) iterations.

Having successfully completed all the aforementioned experiments, someone could observe that the implementation of K-Means running on top of DERBY has a better performance than the other implementation (K-Means running on top of HDFS), when the chosen number of clusters becomes smaller. As a consequence, while the number of the clusters becomes larger and larger, the K-Means implementation on top of HDFS is a better and less cost effective solution for clustering the data. Thus, the total execution time for both the implementations, depends highly on the chosen number of clusters.

As for the final output results, it is clear that both implementations have exactly the same clustering results, as the final clusters' values that they produce have no difference between them. However, both implementations are set to be running the K-Means algorithm for maximum twenty (20) iterations. Because of this, when K-Means reaches this maximum number of iterations, in both cases it is completed and as a result it stops running. Hence, we cannot conclude whether an implementation is better than the other one or not in terms of convergence, as in the experiments that were made, none of the implementations in this maximum number, produced converged clusters as a final result. For that reason, we cannot have a clear view in which implementation the K-Means algorithm runs for less iterations so as to produce the final converged clusters that someone would expect.

Moreover, as someone can observe from the results, the final number of data points that have been assigned to each produced final cluster are pretty much the same for each cluster, meaning that there is not any cluster that has for example five (5) assigned data points, whilst the others have fifty (50) assigned data points and more.

However, it should be mentioned, that all the experiments, were made by changing only a few input parameters of the algorithm, these of the number of the clusters (k) and total the number of the input data points (instances). The distance measure that was used for both of them was the Euclidean distance measure, a measure that if it was changed, the results of the implementations would be completely different. Probably the algorithm would produce better results, with better assigned data points to each cluster, and as a result better final positions of the clusters. Apart from this, both implementations used an 11-dimensional dataset, whose dimensions never changed. So, if the dimensions of the dataset were decreasing or increasing correspondingly, then maybe the output results were quite different. Thus, it becomes clear that there should be done more and more tests on different datasets, with different input parameters for each implementation, so as to have a clearer view on which implementation is better.

Finally, it is worth mentioning that even if the K-Means implementation on top of DERBY using a new distributed key-value pair concept is a prototype project, the final results that it produces do not differ from the results of the original's K-Means implementation on top of HDFS. As a consequence, the new prototype project could be considered as a really good effort, successfully completed. However, it should be mentioned that using the DERBY centralized database instead of an OLAP distributed database was not a random choice, as at a first place we wanted to be sure that the new implemented project runs and outputs correct clustering results. We already know that Hadoop is designed for working with big data, but the aforementioned project is just a prototype, and should be first tested with easy interpretable and understandable datasets. For sure, using DERBY with big data can appear a bottleneck, as DERBY is not designed for that purpose, but the next goal of the project is to test it on top of a scalable OLAP database.

Chapter 7

Conclusions

7.1. Conclusions

After the fulfilment of the Thesis, in a few words, we could say that the overall goals which have been set during the chapter of the Introduction, have been successful. In short, our most important goal, meaning the implementation of the K-Means algorithm on top of an OLAP database using Mahout's machine learning algorithms, was fully completed. We were able to gather valuable results and information after the experiments that have been done, concluding that the new implementation of the clustering algorithm of K-Means, was working properly, having the same or even better results than its older implementation which was running on top of HDFS. In the following paragraphs are stated in deep details the goals that have been set and successfully completed.

To begin with, in this Thesis there were significant and completed information about the Software Systems in general, including the Centralized and the Distributed Systems. It was a very important step for someone to be able to understand the benefits that Distributed Systems can nowadays offer us. Additionally, in order to make it even clearer, there was referred a short comparison between these two kinds of systems.

Our next step, was to fully integrate into the area of the data. It is obvious that huge amounts of data, can be found everywhere, all over the world. Every day we produce thousands of gigabytes of data, creating the need of gaining valuable information and insights by exploring them. For that reason, we studied in deep details about the area of the Knowledge Discovery in Databases, giving more importance to one of its steps, the Data Mining field.

Our goal which has been set from the beginning, was to implement a new clustering algorithm. For that reason, more details were given about the area of clustering, including the techniques and the algorithms that belong to it. However, for implementing a new clustering algorithm, running in a distributed mode, we firstly studied about the distributed framework of Hadoop, which is using distributed computational and storing techniques, to achieve faster and better results, in whichever project it is running. We have seen how it works, we explained the reason of its existence and finally, we talked about its ecosystem, including the projects that are using it in order to achieve their goals.

One of these projects is Mahout, a project which includes Machine Learning libraries in order to perform its clustering, classification or recommender engines' algorithms. After analyzing the most important aspects of Mahout, having in mind that we should understand how it performs its clustering techniques, we studied and gave more details about that last part. We saw how the data should be pre-processed and transformed into vector format, and how in general a clustering algorithm is run, using its Machine Learning libraries.

The clustering algorithm that we chose to implement to run on top of an OLAP database, was the K-Means clustering algorithm. It can be considered as one of the most easy to understand algorithms in general, so it was considered as a good choice, in order to get started. For that reason, we analyzed in deep details how it works and performs its techniques, concluding to how it is implemented on top of HDFS, using the MapReduce paradigm, using Mahout's Machine Learning libraries.

Two separate chapters were given for that reason. The first explained how the K-Means algorithm is using the distributed computational techniques of MapReduce and the HDFS storage. Then, it was time for the implementation to get analyzed. Deep details, including multiple examples were written, making it as less complex as possible, in order to implement it again with a different key-value pair and storing concept.

In general, as we have seen from the chapter with the experiments and the results, there was nothing that the new implementation was missing, comparing it with the original one. Both implementations had exactly the same clustering results, whereas we discovered that when the chosen number of the initial input clusters was smaller, then the clustering results were much faster with the newer implementation.

Having all that in mind, we can say that generally the Data Mining field is something that seems to be very important for our daily lives. It is both very difficult and very challenging to explore that area, but the results that outcome from it, can help us everywhere. Data Mining techniques are used everywhere around us, such as in social networks, in search engines, in web, in e-commerce or dating sites, so this can be a more than enough reason, to make someone interested in that field.

However, we know that data is getting bigger and bigger every minute that is passing by, so more effective and efficient techniques have to be developed, in order to understand their usefulness. In addition, the challenges and the difficulties are augmenting by their side as the databases become bigger, their relationships become more complicated, and the integrity and the security of the data becomes more difficult to be achieved.

Concluding, we see that luckily many projects are being developed nowadays, having as final goal, to exploit in the best way, the various kinds of data which the only thing that have in common is their massive volume and their excessive speed of growth.

The implementation of the K-Means algorithm, was just the beginning for me, as I could say that the Data Mining field, has already captured me. Many tests and experiments should follow, including the improvement of the source code and of the whole distributed idea that is using.

7.2. Future Plans

As for my future plans, as I have already stated, I want to continue exploring the Data Mining field, and find the treasures that data is hiding. Data Mining, and in general the field of Data Analysis, can offer many benefits and challenges, with which I am willing to deal with.

As for the algorithm implementation, we should have in mind that it is a project developed in the last months, and tested in the last weeks. For that reason, for sure it can be said that it has some gaps, including security or integrity gaps, which should be corrected the soonest possible. There are various ways of correcting them, but this will take much time, and various versions of the implementation will have to be created.

The implementation has to be tested several times, and parameterized in order to be able to examine and inspect more kinds of data, and not only numerical non-noisy data. As it has been mentioned, it was designed to communicate only with numerical datasets, but unfortunately, today data varies a lot. Data can be found in messages, in photographs, in videos, in our mobile phones and so on, so there has to be implemented a way to be able to translate all these types of data to a specific readable and interpretable form.

As we have seen from the results, the distributed concept that the K-Means implementation is using instead of the MapReduce paradigm, has given good results, which are exactly the same as the other's implementation. In addition, we saw that sometimes, this implementation performs even better than the other one. However, one of the future plans is to test and to configure the whole implementation on top of an OLAP distributed database, instead of the DERBY database that was currently used for the experiments, using at the same time massive amounts of data.

In addition, as it has been said during the conclusions, we should not stop at the implementation of the K-Means algorithm, as there are numerous of other algorithms, from different categories and families which have to be studied, to be implemented and to be tested, helping in the solution of the data problems that are beginning to appear. So this will be for sure one among my other future plans, to study and try to implement as many algorithms as I can.

All of the above, require years of studies, reading, experiments, failures and efforts which I am willing to spend, as the Data Mining field can be both challenging and full of surprises, whereas it can help us to predict results and make decisions, which will probably solve many of our daily problems.

References

[1] System software, Publicly available from:
http://techterms.com/definition/systemsoftware

[2] System software, Publicly available from:
http://whatis.techtarget.com/definition/system-software

[3] Software System, Publicly available from:
http://en.wikipedia.org/wiki/Software_system

[4] Database System Architectures, Publicly available from:
http://codex.cs.yale.edu/avi/db-book/db4/slide-dir/ch18-2.pdf

[5] Centralized vs Distributed Systems, Publicly available from:
http://www.slideshare.net/zirram/centralized-vs-distrbution-system

[6] Centralized Computing, Publicly available from:
http://en.wikipedia.org/wiki/Centralized_computing

[7] Centralized Computing, Publicly available from:
http://www.techopedia.com/definition/26507/centralized-computing

[8] Centralized vs Distributed Computing, Publicly available from:
http://www.compuquip.com/it-services-blog/2009/11/centralized-vs-distributed-computing/

[9] Distributed Systems Topologies: Part 2, Publicly available from:
http://www.mba.unic.ac.cy/MBA731/DistSysTopologies.pdf

[10] Distributed Systems Architecture, Kay Römer, Frank Pilhofer, Arno Puder, Published by Morgan Kaufmann, 2011

[11] Centralized vs Distributed Systems, Publicly available from:
http://homes.cs.washington.edu/~arvind/cs422/lectureNotes/nw-6.pdf

[12] Database System Architectures, Publicly available from: http://codex.cs.yale.edu/avi/db-book/db4/slide-dir/ch18-2.pdf

[13] Distributed Computing, Publicly available from: http://en.wikipedia.org/wiki/Distributed_computing

[14] Distributed System Principles, Publicly available from: http://www0.cs.ucl.ac.uk/staff/ucacwxe/lectures/ds98-99/dsee3.pdf

[15] Distributed systems at a high level, Publicly available from: http://book.mixu.net/distsys/intro.html

[16] Distributed System, Publicly available from: http://www.cs.gsu.edu/~cscskp/DistSystems/chap01-prasad.pdf

[17] Distributed System Theory, Publicly available from: http://web.info.uvt.ro/~petcu/distrib/SD1.pdf

[18] Distributed System Definitions, Publicly available from: http://students.depaul.edu/~tkyawzaw/ds-definations.html

[19] The Law of Rule: Centralized, Decentralized and Distributed Systems, Publicly available from: http://www2.cffn.ca/usha/part-iii-article-by-pramod-dhakal/129-the-law-of-rule-centralized-decentralized-and-distributed-systems

[20] Data Mining: Introductory and Advanced Topics, M. H. Dunham., Published by Prentice Hall, 2003

[21] Advances in knowledge discovery and data mining, U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, AAAI Press/The MIT Press, 1996.

[22] Data Mining: Concepts and Techniques, J. Han and M. Kamber. Published by Morgan Kaufmann, 2000

[23] Introduction to Knowledge Discovery in Databases, Publicly available from: http://www.ise.bgu.ac.il/faculty/liorr/hbchap1.pdf

[24] Data Mining Definition and Origins, Publicly available from: http://www.dataminingarticles.com/info/data-mining-introduction/

[25] Knowledge Discovery in Databases, Publicly available from: http://www.cise.ufl.edu/~ddd/cap6635/Fall-97/Short-papers/KDD3.htm

[26] KDD - Knowledge Discovery in Databases, Publicly available from: http://seclab.cs.ucdavis.edu/projects/misuse/meetings/KDD.html

[27] Overview of the KDD process, Publicly available from: http://www2.cs.uregina.ca/~dbd/cs831/notes/kdd/1_kdd.html

[28] Knowledge Discovery in Databases, Publicly available from: http://www.usc.edu/dept/ancntr/Paris-in-LA/Analysis/discovery.html

[29] Knowledge Discovery in Databases: An Overview, William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus, Al Magazine Volume 13 Number 3, 1992

[30] Knowledge Discovery in Databases (KDD), Publicly available from: http://www.techopedia.com/definition/25827/knowledge-discovery-in-databases-kdd

[31] From Data Mining to Knowledge Discovery in Databases, Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth, Al Magazine Volume 17 Number 3, 1996

[32] Encyclopedia of Information Science and Technology, Third Editionby Mehdi Khosrow-Pour Published by IGI Global, 2014

[33] Introduction to KDD and DM, Publicly available from: http://minethedata.blogspot.com.es/2012/04/1-introduction-to-knowledge-discovery.html

[34] KDD process, Publicly available from: http://www.cs.helsinki.fi/u/ronkaine/tilome/luentomateriaali/TiLoMe-200105.pdf

[35] Introduction to KDD and DM, Publicly available from: http://minethedata.blogspot.com.es/2012/04/1-introduction-to-knowledge-discovery.html

[36] Predictive Analytics and Data Mining, by Vijay Kotu, Bala Deshpande, Published by Morgan Kaufmann, 2014

[37] Data Mining: Practical Machine Learning Tools and Techniques, 3rd Edition, Eibe Frank, Ian H. Witten, Mark A. Hall, Published by Morgan Kaufmann, 2011

[38] Data Mining and Analysis, Mohammed J. Zaki, Wagner Meira Jr, Published by Cambridge University Press, 2014

[39] Data Mining and Predictive Analysis, Colleen McCue, Published by Butterworth-Heinemann, 2006

[40] Introduction to Data Mining, Osmar R. Zaiane, 1999

[41] Data Mining: Concepts and Techniques, Jiawei Han, Micheline Kamber, Published by Elsevier Science, 2011

[42] http://webdocs.cs.ualberta.ca/~zaiane/courses/cmput690/notes/Chapter1/

[43] Algorithms for Clustering Data, A. K. Jain and R. C. Dubes, Prentice-Hall, 1988

[44] Finding Groups in Data: An Introduction to Cluster Analysis, L. Kaufman and P. J. Rousseeuw, John Wiley & Sons, 1990

[45] B. S. Everitt. Cluster Analysis. Edward Arnold, 1993

[46] Summarization - Compressing Data into an Informative Representation, Varun Chandola, Vipin Kumar, Department of Computer Science, University of Minnesota, MN, USA

[47] Summarization Techniques for Visualization of Large Multidimensional Datasets, Sarat M. Kocherlakota, Christopher G. Healey, Knowledge Discovery Lab, Department of Computer Science, North Carolina State University

[48] Data Mining Association Analysis: Basic Concepts and Algorithms, Publicly available from: http://www-users.cs.umn.edu/~kumar/dmbook/dmslides/chap6_basic_association_analysis.pdf

[49] Association Rules Introductory Overview, Publicly available from: http://www.statsoft.com/Textbook/Association-Rules#overview

[50] Association Rules, Publicly available from: https://courses.cs.washington.edu/courses/csep546/03sp/pdf-slides/9.pdf

[51] Data Mining Classification, Publicly available from: http://courses.cs.washington.edu/courses/csep521/07wi/prj/leonardo_fabricio.pdf

[52] Construction and Assessment of Classification Rules, D. J. Hand, Published by Wiley, 1997

[53] Regression, Publicly available from: http://www.comp.dit.ie/btierney/oracle11gdoc/datamine.111/b28129/regress.htm

[54] Survey of clustering data mining techniques. P. Berkhin, Accrue Software, San Jose, CA, 2002

[55] A Review: Comparative Study of Various Clustering Techniques in Data Mining, Aastha Joshi, Rajneet Kaur, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 3, March 2013

[56] B. S. EVERITT, S. LANDAU, AND M. LEESE, Cluster analysis, Arnolds, a member of the Hodder Headline Group, 2001

[57] Introduction to partitioning-based clustering methods with a robust example, Sami Ayramo, Tommi Karkkainen, Department of Mathematical Information Technology, University of Jyvaskyla

[58] A tutorial on Clustering Algorithms, Publicly available from: http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/

[59] Cluster analysis for applications, M. R. ANDERBERG, Academic Press, Inc., London, 1973

[60] Regression, Publicly available from: http://docs.oracle.com/cd/B28359_01/datamine.111/b28129/regress.htm

[61] Adaptive Control Processes, R. E. Bellman., Princeton University Press, Princeton, NJ, 1961

[62] Automatic subspace clustering of high dimensional data for data mining applications, R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98), pages 94–105, 1998

[63] Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering, A. Hinneburg and D. Keim, 25th International Conference on Very Large Data Bases (VLDB'99), Morgan Kaufmann, 1999

[64] Finding generalized projected clusters in high dimensional spaces, C. C. Aggarwal and P. S. Yu., 2000 ACMSIGMOD International Conference on Management of Data (SIGM'99), volume 29(2), ACM Press, 2000

[65] Spatial Clustering Methods in Data Mining: A Survey. J. Han, M. Kamber, and A. K. Tung, In Miller, H. and Han, J. Geographic Data Mining and Knowledge Discover. Taylor and Francis, 2001

[66] Data clustering: a review, A. K. Jain, M. N. Murty, and P. J. Flynn. ACM Comp. Surveys, 1999

[67] Some methods for classification and analysis of multivariate observations, J. MacQueen, Fifth Berkeley Symposium on Mathematical Statistics and Probability. Volume I,Statistics., 1967

[68] Efficient and Effective Clustering Methods for Spatial Data Mining, R. T. Ng and J. Han., 20th VLDB Conference, pages 144–155, Santiago, Chile, 1994

[69] BIRCH: An Efficient Data Clustering Method for Very Large Databases, Tian Zhang, Raghu Ramakrishnan, and Miron Livny., 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Canada, 1996

[70] Cure: An efficient clustering algorithm for large databases, Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim., ACM SIGMOD conference on Management of Data, Seattle, WA, 1998

[71] Rock: A robust clustering algorithm for categorical attributes, S. Guha, R. Rastogi, and K. Shim., IEEE Conference on Data Engineering, 1999

[72] Improved Outcome Software, Agglomerative Hierarchical Clustering Overview, Publicly available from: http://www.improvedoutcomes.com/docs/WebSiteDocs/Clustering/Agglomerative_Hierarchical_Clustering_Overview.htm

[73] Data Mining: Concepts and Techniques, Han, J., Kamber, M. 2012., 2nd edition, Published by Morgan Kaufmann, 2006

[74] A density-based algorithm for discovering clusters in large spatial databases with noise, M. Ester, Kriegel H.-P., J. Sander, and X. Xu., 2nd International Conference on Knowledge Discovery and Data Mining, 1996

[75] An efficient approach to clustering in large multimedia databases with noise, Alexander Hinneburg and Daniel A.Keim., Fourth International Conference on Knowledge Discovery and Data Mining, New York, August 1998

[76] OPTICS: Ordering Points To Identify the Clustering Structure, Ankerst M., Breunig M. M., Kriegel H.-P., Sander J., ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'99), Philadelphia, PA, 1999

[77] Automatic subspace clustering of high dimensional data for data mining applications, Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan, ACM SIGMOD Conference on Management of Data, Seattle, WA, 1998

[78] Wavecluster: A multi-resolution clustering approach for very large spatial databases, G. Sheikholeslami, S. Chatterjee, and A. Zhang, 24th International Conference on Very Large Data Bases, 1998

[79] STING: A Statistical Information Grid Approach to Spatial Data Mining, Wei Wang, Jiong Yang, and Richard Muntz., 23rd VLDB Conference, Athens, Greece, 1997

[80] Data Clustering Techniques, Periklis Andritsos, Department of Computer Science, University of Toronto, 2002

[81] An Overview on Clustering Methods, T. Soni Madhulatha, Alluri Institute of Management Sciences, Warangal, IOSR Journal of Engineering, Vol. 2(4), April 2012

[82] Hadoop in Action, Chuck Lam, Published by Manning Publications, 2013

[83] Hadoop Overview, Publicly available from: http://www.revelytix.com/?q=content/hadoop-overview

[84] Hadoop in Practice, Alex Holmes, Published by Manning Publications, 2012

[85] Hadoop-Beginner's Guide, Garry Turkington, Published by Packt Publishing, 2013

[86] Introduction to Apache Hadoop, Publicly available from: http://binarynerd.com/java-tutorials/distributed-computing/intro-apache-hadoop.html

[87] The history of Hadoop: From 4 nodes to the future of data, Publicly available from: https://gigaom.com/2013/03/04/the-history-of-hadoop-from-4-nodes-to-the-future-of-data/2/

[88] The History of Hadoop: From Small Starts to Big Data, Publicly available from: http://blog.pivotal.io/pivotal/news-2/the-history-of-hadoop-from-small-starts-to-big-data

[89] An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics, Ronald C Taylor, 11th Annual Bioinformatics Open Source Conference (BOSC) 2010, Boston, MA, USA. 9-10 July 2010

[90] Hadoop for Dummies, Dirk deRoos, Published by For Dummies, 2014

[91] An introduction to Apache Hadoop for big data, Publicly available from: http://opensource.com/life/14/8/intro-apache-hadoop-big-data

[92] Apache Hadoop, Publicly available from: http://en.wikipedia.org/wiki/Apache_Hadoop#History

[93] Apache Hadoop core components, Publicly available from: http://docs.hortonworks.com/HDPDocuments/HDP1/HDP-1.2.4/bk_getting-started-guide/content/ch_hdp1_getting_started_chp2_1.html

[94] Hadoop Ecosystem, Publicly available from: http://thinkbig.teradata.com/leading_big_data_technologies/hadoop/

[95] Hadoop Design, Architecture & MapReduce Performance, Publicly available from: http://www.datanubes.com/mediac/HadoopArchPerfDHT.pdf

[96] Hadoop architecture, Publicly available from: http://hadooptutorial.wikispaces.com/Hadoop+architecture

[97] Introduction to Big Data & Hadoop Ecosystem – Part 1, Publicly available from: http://yourstory.com/2012/04/introduction-to-big-data-hadoop-ecosystem-part-1/

[98] HADOOP FILE SYSTEM AND FUNDAMENTAL CONCEPT OF MAPREDUCE, A.Pradeepa, Antony Selvadoss Thanamani, International Journal of Advanced Research in Computer and Communication Engineering, Vol. 2, Issue 10, October 2013

[99] Introduction to Big Data & Hadoop Ecosystem – Part 2, Publicly available from: http://yourstory.com/2012/04/introduction-to-big-data-hadoop-ecosystem-part-2/

[100] Hdfs architecture, Publicly available from: http://hadoop.apache.org/common/docs/current/hdfs_design.html

[101] The Hadoop Distribyted File System, Dhruba Borthakur

[102] Hadoop Ecosystem, Publicly available from:
http://www.revelytix.com/?q=content/hadoop-ecosystem

[103] Introduction To Parallel Programming And MapReduce, 2007 Google

[104] Apache Hadoop, Publicly available from:
http://hadoop.apache.org/common/docs/current/mapred_tutorial.html

[105] Hadoop MapReduce Cookbook, Thilina Gunarathne, Srinath Perera, Published by Packt Publishing, 2013

[106] Hadoop animal planet, Publicly available from:
http://blogs.hexaware.com/big-data/hadoop-animal-planet/

[107] Hadoop: Open Source Big Data Analytics Tools, Publicly available from:
http://blog.qburst.com/2014/08/hadoop-big-data-analytics-tools/

[108] Ecosystem of Hadoop Animal Zoo, Publicly available from:
http://j2eedev.org/ecosystem-hadoop-animal-zoo/

[109] Hadoop 101: An Explanation of the Hadoop Ecosystem, Publicly available from:
http://architects.dzone.com/articles/hadoop-101-explanation-hadoop

[110] Understanding Hadoop Ecosystem, Publicly available from:
http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.5.0/bk_getting-started-guide/content/ch_hdp2_getting_started_chp2.html

[111] Hadoop: The Components You Need to Know, Publicly available from:
http://dataconomy.com/hadoop-components-need-know/

[112] Hadoop 101: An Explanation of the Hadoop Ecosystem, Publicly available from:
http://architects.dzone.com/articles/hadoop-101-explanation-hadoop

[113] Introduction to Big Data & Hadoop Ecosystem – Part 3, Publicly available from:
http://yourstory.com/2012/04/introduction-to-big-data-hadoop-ecosystem-part-3/

[114] Applicability of MAHOUT for Large Data Sets, Michael Sevilla, University of California, Santa Cruz

[115] Apache Mahout, Publicly available from: http://alternativeto.net/software/apache-mahout/

[116] An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics, Ronald C Taylor, 11th Annual Bioinformatics Open Source Conference (BOSC) 2010, Boston, MA, USA. 9-10 July 2010

[117] Getting Started with Apache Mahout, Publicly available from: http://technobium.com/getting-started-with-apache-mahout/

[118] Distributed Analytics: A Primer, Publicly available from: http://thomaswdinsmore.com/tag/apache-mahout/

[119] Introducing Apache Mahout, Publicly available from: http://www.ibm.com/developerworks/java/library/j-mahout/

[120] Big Data Glossary, Pete Warden, Published by O'Reilly Media, Inc., 2011

[121] Mahout in Action, Ted Dunning, Sean Owen, Robin Anil, and Ellen Friedman, Published by Manning Publications, 2011

[122] Apache Mahout Cookbook, Piero Giacomelli, Published by Packt Publishing, 2013

[123] Using a recommendation engine to personalize your web application, Publicly available from: http://www.ibm.com/developerworks/websphere/techjournal/1109_zegarra/1109_zegarra.html

[124] Introduction to Apache Mahout, Publicly available from: http://www.javawebdevelop.com/2491740/

[125] Cluster Dumper – Introduction, Publicly available from: https://mahout.apache.org/users/clustering/cluster-dumper.html

[126] Introducing Apache Mahout, Publicly available from: http://www.ibm.com/developerworks/java/library/j-mahout/

[127] Introduction to Clustering in Mahout, Publicly available from: http://www.edureka.co/blog/introduction-to-clustering-in-mahout/

[128] Cluster Analysis: Basic Concepts and Algorithms, Publicly available from: http://www-users.cs.umn.edu/~kumar/dmbook/ch8.pdf

[129] Efficient Disk-Based K-Means Clustering for Relational Databases, Carlos Ordonez, Edward Omiecinski, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 16, NO. 8, AUGUST 2004

[130] K-means Clustering, Publicly available from: http://databases.about.com/od/datamining/a/kmeans.htm

[131] K-Means Clustering Tutorial, Kardi Teknomo, July 2007

[132] K-Means Clustering, Publicly available from: http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html