

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 FUZZY K-MEANS CLUSTERING ON TOP OF AN OLAP DATABASE

Master Thesis

By

Athanasios Kiourtis

Supervisors: Ricardo Jiménez Peris
Andriana Prentza

Madrid, 13/02/2015

The undersigned have examined the thesis entitled "**Parallel Fuzzy K-Means Clustering on top of an OLAP Database"** presented by **Athanasios Kiourtis**, 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

Αφιερωμένο στην μαμά μου, στον μπαμπά μου, στην αδερφή μου και στην κοπέλα μου…

(Dedicated to my mom, my dad, my sister and my girlfriend…)

Acknowledgements

I would like to express my gratitude to Ricardo Jiménez Peris, Marta Patiño-Martínez and Andriana Prentza for their useful comments, remarks and engagement through the learning process of this master thesis. Their contributions not only to my study, but also to my academic vision made me excited and hopeful for further studies.

Furthermore, I would like to thank the members of the committee, María Pérez-Hernández and Javier Soriano, for their comments, their patience and their time.

Special thanks to Dimos Kyriazis, as without him, coming to the UPM as an Erasmus Student would be just a dream.

I want to thank my Spanish friends Ivan, Alejandra, Valerio and Ainhoa, for making things funnier and helping me with their language skills, during these months.

Thank you Louli for being there to help me and support me, no matter time or day of the week, both as a friend and as a partner.

Last but not least, many thanks to my mother (Anna), my father (Dimitris) and my sister (Asimenia), as without them I would not be able to go any further. I will be grateful forever.

Σας ευχαριστώ,
Θάνος

Abstract

The era we live in is attacked by the "big data" phenomenon. Multiple enterprises store large amounts of data for analysis, making the field of data analysis more and important. But how is someone able to gain insights from this massively evolving data? There comes the data mining field, which is what this thesis investigates, and especially the clustering field, whose algorithms can group unknown data into multiple clusters.

Needing to perform clustering in a distributed way, the framework of Hadoop is analyzed, which offers a way to execute parallel programs on a cluster of machines, storing the data in a distributed file system. The parallel executions are achieved using the MapReduce paradigm which transforms complex computations over a set of <key, value> pairs, so that many jobs can run together. As for the distributed storing, it is achieved using the Hadoop distributed file system (HDFS), a file system that provides scalable and reliable data storage. One of the projects that uses Hadoop for performing clustering, classification and collaborative-filtering techniques to large data, is Mahout.

Among all the clustering algorithms, the thesis gives details about a Fuzzy clustering algorithm, Fuzzy kMeans, which groups the data to clusters, by assigning to each data point different degrees of association for each cluster.

The main problem that has to be solved is how we can cluster data stored in an OLAP database, in a parallel way. For that reason, it is studied how Mahout implements the fuzzy kMeans algorithm using the Hadoop's components, and after that, the thesis proposes a "similar approach" of the algorithm, running on top of an OLAP database.

In more details, the new fuzzy kMeans implementation instead of MapReduce is using a three-stepped <key, value> pair idea (Map, Reduce & FinalReduce). In the first step, the clustering jobs are split and assigned to different threads which export their own <key, value> pairs. In the second step, each thread, without waiting for the other threads to finish their first step, continues using the extracted <key, value> pairs from its previous step and exports its own <key, values>. The last step, takes place when all of the threads finish the aforementioned two steps, and the results of the second step are merged together, to produce the final clustering result. In addition, instead of using the HDFS for storing purposes, the implementation is making use of an OLAP database.

For the previous idea, a prototype implementation has been developed and preliminary tests were run, comparing the different fuzzy kMeans clustering implementations.

Finally, the thesis concludes with the fact that more attention should be given to the data that evolve day by day, as mining and extracting information out of it, becomes more and more complicated, whereas it could be considered as a very challenging and attractive job.

Resumen

Actualmente estamos viviendo un momento que está sufriendo el fenómeno "big data". Muchas empresas almacenan enormes cantidades de datos para su posterior análisis, haciendo que el campo del data mining esté adquiriendo mayor importancia. Pero, ¿cómo puede ser alguien capaz de obtener información de estos datos que evolucionan de forma masiva? Es aquí donde entra en juego el campo del data mining, sobre el cual se centra el estudio de esta tesis y en especial el campo del clustering, cuyos algoritmos permite agrupar datos que a simple vista no tienen relación entre ellos.

Debido a la necesidad de ejecutar estos algoritmos de clustering de manera distribuida, se va a utilizar Hadoop, un sistema distribuidos que ofrece la posibilidad de ejecutar programas en paralelo sobre un conjunto de máquinas. La ejecución paralela se consigue mediante la utilización del paradigma MapReduce el cual transforma computaciones complejas sobre un conjunto de pares clave-valor de tal manera que diversas tareas puede ser ejecutadas conjuntamente. Para conseguir el almacenamiento distribuido de los datos, se está utilizando HDFS, un sistema que provee escalabilidad y fiabilidad al almacenamiento de los datos. Uno de los proyectos que utiliza Hadoop para la ejecución de técnicas de clustering, clasificación y filtrado colaborativo sobre grandes cantidades de datos es Mahout.

Entre todos los algoritmos de clustering existentes, en esta tesis se ha utilizado un algoritmo fuzzy, en concreto Fuzzy k-means, el cual agrupa los datos en grupos mediante la asignación a cada dato de un grado de asociación con cada uno de los grupos.

El principal problema a resolver es como se puede aplicar el algoritmo de clustering sobre una base de datos OLAP de manera paralela. Por esta razón, se ha estudiado como Mahout implementa el algoritmo Fuzzy K-Means utilizando los componentes que ofrece Hadoop y a continuación proponer aproximaciones similares del algoritmo que permitan ser ejecutadas sobre una base de datos OLAP.

Concretamente, la nueva implementación del algoritmo Fuzzy K-Means utiliza, en vez de MapReduce, un concepto de pares clave-valor basado en tres pasos (Map, Reduce y FinalReduce). En el primero paso, el proceso de clustering es dividido y asignado a diferentes procesos los cuales exportan sus propios pares clave-valor. En el segundo paso, cada proceso, sin esperar al resto de procesos a que terminen el primer paso, continúan utilizando los pares clave-valor del paso anterior y exportan son propios pares clave-valor. El último paso tiene lugar cuando todos los procesos han terminado los dos pasos anteriores y los pares clave-valor resultantes del segundo pasos de todos los procesos son integrados, dando lugar al resultado final del clustering. Además, en vez de utilizar HDFS como sistema de almacenamiento, se ha utilizado una base de datos OLAP.

De esta manera, se han realizado diversas implementaciones del algoritmo de clustering Fuzzy K-Means, para realizar posteriormente distintos experimentos que han permitido la comparación entre ellos.
Finalmente, se ha concluido que es necesario prestar mayor atención a los datos que evolucionan constantemente, así como a la extracción de información ya que este proceso cada vez es más complicado, dando lugar a ser considerado como una tarea desafiante y atractiva.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Summary: "*In this chapter it is given an analysis of the goals and the purposes of the following thesis, including a short description of the problems that we have to deal with. Additionally, the structure of thesis is written, explaining the way the whole thesis has been split, beginning from the study of the state of the art, to the conclusions and the future plans.*"

1.1 Goals – Purpose of the thesis

The era we live in consists of the evolution of information systems, which has driven on the creation of a society which deals with various types and large amounts of information. This kind of information is stored in a continuous way, and as a result of this, the data gets bigger and bigger creating the need to exploit it with multiple complex ways, to gain valuable insights.

The data has been replaced by the big data phenomenon, as the companies and the firms, have databases which can store almost everything. The problem that arises is how can this data be processed, and how can someone gain insights, retrieve information and make predictions, by the time that it needs so much time and expensive hardware, to analyze hundreds of petabytes of data?

Various ways of exploiting big data have been introduced and are nowadays used, but each of these ways have both advantages and disadvantages, while they can sometimes be ineffective. In addition, in order to manage and analyze this data, the computer science has been driven into the creation of the data mining field, which includes a series of techniques based on various algorithms.

Many projects have been created in order to read and analyze data simultaneously, and are trying to cooperate with each other, in order to have the best results. Projects like Hadoop, technologies like MapReduce and sub-projects like Mahout, are combined with the most effective way, in order to help the companies and the firms, to have faster, better and more valuable results.

For that reason, in this thesis there are going to be explained some of the technologies and projects that have been developed to analyze and work with big data, which use various data mining techniques for gaining valuable insights.

In general, there will be a study of the state of the art on the field of data analysis, and after that the framework of Hadoop will be explained, which referring to the data analysis fields, it is used to achieve distributed data analysis results. The MapReduce paradigm and the Hadoop distributed file system (HDFS) will be explained, including the subproject of Mahout, which uses the core components of Hadoop to perform data mining techniques on massive data sets, in a high-scalable way.

In addition, a new clustering algorithm implementation will be developed, using techniques and technologies that are going to be introduced. In more details, the main problem that we are dealing with in that thesis is how we can perform parallel clustering techniques in large amounts of data that are stored in an OLAP database. Until today, in order to achieve parallel clustering on massive data, the project of Mahout has been developed which is making use of MapReduce, a batch processing based concept, which splits the data into <key, value> pair format, for achieving parallel results. But what we need and we will develop in that case is a concept for clustering data that is stored in an OLAP database, in a parallel way.

The new implementation will split again the clustering job into smaller parts, by assigning a different part of the job to different threads. The <key, value> pair idea will still be used, but instead of waiting for the Map tasks to be completed in order to continue with the Reduce tasks, each thread will perform its corresponding Map and Reduce tasks, without waiting for the other threads to finish their jobs. At the very end, when all the threads finish working, their results will be merged together.

We know that data is not static, so having results in real-time has a great importance, as this could help enterprises in understanding their customers' needs, making decisions or dealing better with difficult situations.

Having developed the clustering algorithm to output results in parallel on top of an OLAP database, the next goal is to test it with various parameters and data sets, in order to compare it with the implementation that already exists and is running with Mahout on top of Hadoop.

More details will be given to the following chapters, including the conclusions and the future goals which derive after the finalization of that thesis.

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

- Research and study of the *state of the art on data analysis*.
- Detailed study about what *big data* is and what are the challenges.

- Research of the *knowledge discovery in databases* (KDD) area and the *data mining* field.
- Detailed study of the *clustering* techniques, accompanied with the applications of clustering, its techniques and its algorithms.
- Research of the distributed processing framework of *Hadoop*.
- Understanding the *MapReduce* paradigm and the *Hadoop distributed file system*.
- Study of the *Mahout*, a subproject of the Hadoop's ecosystem, and the way its clustering techniques are implemented.
- Thorough research about the *fuzzy kMeans* clustering algorithm and how it is implemented with Mahout on top of Hadoop.
- Development and explanation of a different parallel implementation of the *fuzzy kMeans* Clustering algorithm on top of an OLAP database, based on a distributed <key, value> pair concept.
- Run *tests* and evaluate the results of the implementations.
- Quote of the *conclusions* which derive from the completion of the thesis, accompanied with the future plans based on the research and the implementation that has been done.

1.2 Structure of the thesis

The thesis, consists of seven (7) chapters. In the beginning of its chapter, there is a short summary which explains in short, the content of each chapter.

More particularly, the chapters are separated as following:

- In *Chapter 1*, there is an introduction about the main subject of the thesis and there are explained the main goals that have been set.

- In *Chapter 2*, takes place the study of the state of the art about the fields that someone has to know, in order to be able to understand the whole idea of the thesis.

- In *Chapter 3*, the Hadoop distributed processing framework is described, in combination with its main components and its ecosystem.

- In *Chapter 4*, the Mahout software is described, in combination with how it implements the clustering techniques.

- In *Chapter 5*, the different implementations of the fuzzy kMeans Clustering algorithms are described, accompanied with detailed figures.

- In *Chapter 6*, the tests and the results are written down, in order to continue with the evaluation of the different clustering implementations.

- In *Chapter 7*, there are described the final conclusions and the future plans after the research and the implementation that has been made.

Chapter 2

State of the art on data analysis

Summary: *"In this chapter, a thorough research is done on the field of the data analysis. There are given various definitions about the area of data analysis, including the big data phenomenon. In addition, the knowledge discovery in databases is explained, accompanied with one of its steps, the data mining field. As for the last field, our interest remains at the clustering techniques and algorithms, as the final goal of the thesis, is to implement a new clustering algorithm."*

2.1 Data analysis

For the last years, it could be said that businesses actually sit on data, and every second that passes, they generate more and more. Data, nowadays, is being produced at faster rates due to the explosion of information which is related to the internet and the use of more and more systems. For sure, there is a way with which we can make use of all this data, but the problem that occurs is "how" this can happen.

The aforementioned seems to be difficult as there is no specific problem that needs to be solved and there is no specific question that needs to be answered. The only information that we have in our hands is the data, whilst our final goal is to improve the business.

The increasing volume of data, the complexity that arises from the different information types that are collected, the reliability and the integrity of the data collected, can be considered as a very difficult challenge.

We can find data all around us, in different types and conditions, in hospitals, in banks, in weather records, in photo albums, in videos, or even in our mobile phones. In fact, data can be seen as the essential raw material of any kind of human activity. Almost every sector from biology and economics to engineering and marketing measures, gathers, and stores data in different kinds of digital form. Insurance companies store information of insurance claims, meteorological organizations measure and store data as for the weather conditions, and banks gather information concerning the personal data of their clients.

For that reason, clever and timely decisions need to be made using the information collected, which will be used in order to improve sales, make better researches and project developments, while reducing the total costs. For instance, insurance companies need to find and claim all the activities which are considered as fraudulent and

meteorological organizations need to be able to predict weather conditions of the near future.

So, in order to solve and simplify the needs mentioned before, a new science raised, the science of *data analysis*. In short, *data analysis* is the process in which raw data is ordered and organized, to be used in methods that help into the explanation of the past and the prediction of the future. It can be considered as a multi-faceted process for the inspection, the cleaning, the transformation and the modeling of the data with the main goal of discovering useful information, suggesting conclusions, and supporting decision-making. Usually, during the final step of the analysis, the data is converted into meaningful information, necessary to take decisions [1], [2], [3], [4].

2.1.1 Data analysis domains

Data analysis does not have to do only with results and numbers, it is about asking questions, developing explanations, gathering information and testing hypotheses [2], [4]. It can be considered as a multi-disciplinary sector, which combines the domains of:

- Computer science,
- Artificial intelligence & machine learning,
- Knowledge
- Statistics & mathematics

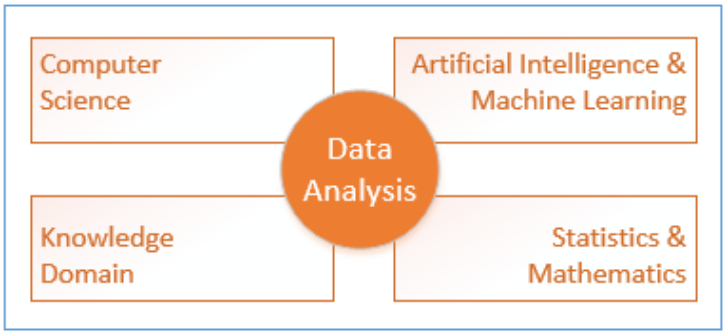

Figure 1 - Data analysis domains

In Figure 1, we can see how data analysis is combined with the aforementioned domains. In a few words, these domains could be explained as follows:

- *Computer science* is responsible for the creation of the various data analysis tools. The large amount of data has made computational analysis very important and has increased the requirements for programming, database and network administration skills.

- *Artificial intelligence* is required for the study of the algorithms that can simulate an intelligent behavior, so it is used in order to perform those activities that require this kind of intelligence.

- *Machine learning* is the study of computer algorithms in order to learn about how to react in a certain circumstances or recognize patterns. According to how the algorithm is training, it consists of a large amount of algorithms generally, categorized in the following categories:
    - Supervised learning (data is labeled)
    - Unsupervised learning (data is unlabeled)
    - Semi-supervised learning (data is both labeled and unlabeled)

- *Statistics* have to do with the development and the application of different methods in order to collect, to analyze, to understand and to translate data. During the phase of data analysis, a variety of statistical techniques are taking place such as clustering, classification, forecasting, prediction and regression.

- As for the *knowledge domain*, a good understanding of it can give to someone the capability to ask good questions, which has a great importance in the field of data analysis. As mentioned earlier, data can be found everywhere, so the information gathered from the data, turn into a set of rules and can help us to make decisions.

- In the field of *mathematics*, data analysis uses lots of mathematical techniques in the using algorithms such as linear algebra, numerical methods and probabilities.

2.1.2 Data categories

Data can be separated into two different categories [5]: *categorical* and *numerical*.

- *Categorical* data are values that can be sorted into different groups or categories. There are two types of categorical values, *nominal* and *ordinal*.
    - A *nominal* variable is the one which has no numerical value (e.g. gender or occupation.)
    - An *ordinal* variable is the one whose order is significant, but on which no meaningful arithmetic operations can be performed (e.g. the survey question "Is your general health poor, reasonable, good, or excellent?" may have those answers coded respectively as 1, 2, 3, and 4.)

- *Numerical* data are values that can be measured. There are two types of numerical values, *discrete* and *continuous*.

- A *discrete* variable can be counted and is always distinct and separate (e.g. the result of a flipped coin.)
- A *continuous* variable can take on any value within a finite or infinite range of numbers (e.g. the weight of average heighted people which should be between 70-80 Kg)

In Figure 2, a simple example is presented of how data can be turned into knowledge:

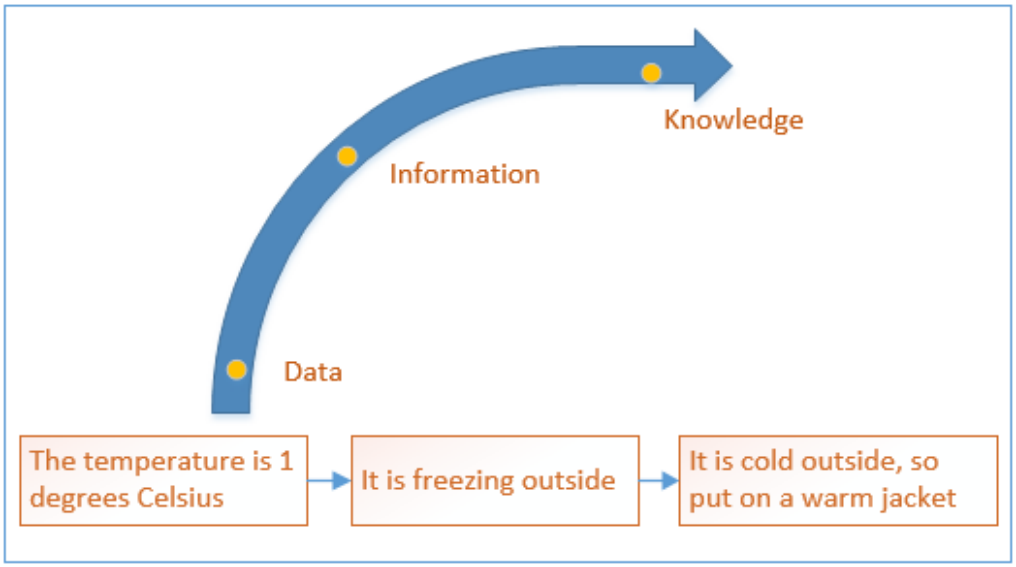

Figure 2 - Transformation of data to knowledge

- As we can see, the *data* that we have in our possession is that:
  "The temperature outside is 1 degrees Celsius".

- As a result of this, after some processing, we gather the *information* that:
  "It is freezing outside".

- So, the *knowledge* that derives from that short data analysis is that we should:
  "Put a warm jacket".

Data analysis however, is not so simple.

2.1.3 Steps of the data analysis process

In the case that we had to deal with much more complicated and bigger data than the one mentioned in the previous example (section 2.1.2), the data analysis process in general consists of several steps which someone should carefully think through. In general, the steps that someone should follow for an effective data analysis can be the following:

- State the problem
- Obtain the data
- Clean the data
- Transform the data
- Explore the data with statistics and visualizations
- Implement predictive modeling
- Visualize and interpret the results
- Deploy the solution

Figure 3, summarizes the data analysis process, including the aforementioned steps. Although it is usual to follow the order described, there are usually interactions between the different steps that may require to be revised. For example, it may be necessary to return to the data preparation while implementing the data analysis in order to make changes based on what is being learned.

Figure 3 - Data analysis process

These steps, are summarized here and explained afterwards:

- (1) *Problem Definition and Planning*: The problem that has to be solved is clearly defined, and an appropriate team is assembled to perform the analysis.
- (2) *Data preparation*: Before the starting of a data analysis project, the data must be gathered, characterized, cleaned, transformed and partitioned into an appropriate form for further processing.
- (3) *Analysis*: Based on the information from the previous steps, the corresponding and appropriate techniques must be selected in order to gain the best results.
- (4) *Deployment*: The results from the previous step should be examined and deployed to obtain what was identified at the beginning of the process.

1) Problem definition and planning

The first step of the data analysis is to describe the problem that has occurred and create a plan to solve it.

During the first phase, the following issues should be taken into consideration:

- Identify the problem
- List the project's deliverables
- Generate success factors
- Understand each resource and find the limitations
- Find an appropriate team
- Generate a plan
- Perform a cost/benefits analysis

It is important to document the problem that has to be solved along with corresponding information. In certain occasions, however, it may not be possible to know precisely the sort of information that will be gathered after the analysis.

In addition, concerning what resources are available for use in the project, there may be limitations on the available data, the hardware or the software that can be used. Also, there could be issues (privacy, legal issues) related to the data usage, which should be identified and written in prior. Moreover, there may also be time limitations for an algorithm to create the final results.

As for the teams, having good knowledge is essential and it is more than helpful to have different roles, as different people will have an active role at different times, so it is desirable to involve all parties during this phase.

Finally, a plan about how the team and the whole analysis process is going to work has to be created, in combination with a cost/benefit analysis, in order to see if the information that will be derived from the analysis would be more valuable than costly.

2) Data preparation

The second step of the data analysis has to do with the understanding and the preparation of the data, to get it ready for Analysis. However, it is the most time-consuming step in the process, since the data is usually gathered from different data sources, with different representations or formats.

During the second phase, the following steps are required for preparing the data set:

- Access and combine data sets
- Summarize the data
- Look for errors and noisy data
- Transform the data
- Partition the data

In general, in occasions where the data has been collected for a different reason, it has to be transformed into an appropriate form for analysis. Since multiple sources of data may be used, it must be taken much care in order not to deal with errors when these sources are combined with each other.

It is important to characterize the attributes types that have been collected over the different items in the data set, so to identify unexpected values. For example, in looking at the numeric attribute weight collected for a set of people, if an item has the value "low" then we need to either to replace it with a numerical value or remove the entire record for that person.

Moreover, it may be important to transform the data to make it more capable for data analysis, which should be done without losing any valuable or important information. Last but not least, the data has to be partitioned in smaller segments, in order to be more easily accessed and processed, with faster and more effective results.

3) Analysis

The third step during the data analysis has to do with the examination of the data, an important step for understanding the type of information that has been collected and the meaning of the data. In combination with the gathered information from the problem definition phase, the correct data analysis approach will be selected.

During the third phase, the following approaches should be considered when analyzing the data:

- Summarize the data
- Explore the relationships between attributes
- Group the data
- Identify non-trivial facts, patterns and trends

First of all, data must be interpreted and summarized, without losing valuable or important information. In addition, there should be a number of tasks for focusing on finding important facts and relationships in the data. Discovering this information, often has to do with looking at the data in many different aspects, using techniques of data visualization, data analysis and data mining.

Furthermore, during the third phase there should be some tasks which have to do with the development of mathematical models that give relationships to the data. Models like these can be useful for understanding the data and for making predictions.

Moreover, a mandatory fact can be considered the selection of the methods to be used, which are often driven by the type of data being analyzed and the problem being solved. Some approaches generate solutions that are very easy to interpret and explain important problems, while others are more limited for explaining the various results.

Finally, understanding hidden relationships between different items in the data can help in generating models. For that reason, it is essential that the data analysis team works closely while analyzing and interpreting the data.

4) Deployment

In the fourth step, the data analysis results are translated into valuable information for the organization. For that reason, this step should be carefully planned and executed, and as a result of that, there are many ways to deploy the output of a data analysis process.

During the fourth phase, the results can be deployed in different ways, as we can see below:

- Generate a report
- Deploy a decision-support tool
- Measure business impact

As for the first option, the data analysis team could write a report describing the business, the results or the scientific intelligence derived from the analysis. Afterwards, the report should be directed to those who are responsible for making decisions.

However, in the case when the results of the process include the generation of predictive models, these models can be deployed as standalone applications or can be integrated with other software. The integration of the results into existing systems is often one of the most expensive approaches to delivering a solution.

Finally, at the end of a project, it is always a useful exercise to look back at the data analysis impact, and to determine what worked and what did not work, as this will provide results, information and ideas for the future improvement of similar kinds of processes [3], [4], [5].

2.2 Data preparation (data cleaning)

There are many different sources of data as well as methods used to collect the data. For instance, surveys or polls can be considered as a common way for gathering data, in order to answer specific questions. Another example could be an interview where a set of questions is used, for gaining information on people's opinions, preferences, capabilities or behavior. However, in the survey there should be selected a random sample of the target population or during an interview there should not be any questions that favor a particular response.

In addition, the data collected should be reliably measured, which means that if the measurement would be repeated then it should not result in different values or results.

Generally, data can be found in many different forms.

By the time that the data is collected, the most time-consuming part of the data analysis has to take part: the *data preparation*. The way in which the data is collected and prepared is critical for the final results, and for that reason it is a very important step.

Most of the times, the data needs to be combined or merged into a table. After that, the data should be cleaned by resolving errors, removing noisy data and deleting columns of irrelevant data. Finally, during the data preparation, the table should be divided, into smaller parts so to make the data analysis clearer and to allow specific questions to be answered more easily.

It should be mentioned that it is important to write down the details about the steps taken and the reason of why they were done, in order to provide a methodology to apply to similar data sets in the future.

Subsequently, the process of preparing the data (data cleaning) for analysis is defined in more details as for the phases of how to obtain, clean, normalize and transform raw data into a standard format.

In order to avoid "dirty" data, the data set that is going to be used should have the following characteristics:

- Be correct
- Be complete
- Be accurate
- Be consistent
- Be uniformed

Some data cleaning tasks include record matching, deduplication and column segmentation. For some variables it is useful to inspect all of their possible values to reveal and correct mistakes, duplications and errors, as each value has a unique term. Another common problem with numeric variables is that sometimes they are written as non-numeric terms (e.g. "below zero"), so these terms should be transformed into number format or removed.

A different problem arises when observations for a particular variable are missing data values, so in that case the value may be replaced according to the knowledge of how the data was gathered in general.

Moreover, during the data cleaning it can be more difficult to clean variables measured on a ratio scale since they can take any possible value within a given range. In that case, it is useful to consider outliers (small number of data values that differ greatly from the rest of the values) in the data.

Finally, it should be taken into consideration that a particular variable may have been measured over different units (e.g. km, miles) and that when data is combined from multiple sources, it is more likely that it have been recorded more than once. In these cases, the data should be converted and duplicate entries should be removed, as well.

Some of the data preparation techniques, are described below, in order to prepare the data for the data analysis process [3], [4].

- Converting text to numbers

In order to use variables that have been assigned as nominal or ordinal and described in text values, it is mandatory for these values to be converted into numerical values. A common way to handle nominal data, is to convert the values into a separate column where values with number 1, indicate the presence of a certain category and values with number 0, indicate the absence of a certain category.

- Converting continuous data to categories

The conversion of continuous data to categories is more desirable when a value is defined on a ratio scale, but there is less knowledge about how the data was collected. For example, a variable weight that has a range from 0 to 100 kg may be divided into five categories: (1) less than 20 kg, (2) 20–40 kg, (3) 40–60 kg, (4) 60–80 kg and (5) 80-100 kg. Afterwards, all values for the variable weight must be assigned to a category and be given an average value of the assigned category.

- Combining variables

Sometimes, the variable that is going to be used may not be present in the data set but it may be a result from some existing variables. In that case, mathematical operations, such as average or sum, could be applied to one or more variables in order to create an extra variable.

- Generating Groups

In general, larger data sets take more time to be analyzed. One solution for that problem could be to take a random part for analysis, but this is only effective when the data set matches the target population. One reason of generating groups, is that dividing the data set into smaller parts based on some knowledge of the data, may allow someone to generate several simpler, faster and more effective models.

2.3 Big data

The question that arises nowadays is, what exactly big data is. Referring to this kind of data, someone could guess that it is something that is large and full of information. More often, big data is described as extremely large data sets that grow so fast that their management and analysis cannot be done using traditional data processing tools. In other words, the data set has become so large that it is so hard to gain value out of it.

In general, big data is a word, used to describe a massive volume of both structured and unstructured data that is so large, making it difficult to process using traditional database and software techniques. In most of the cases, *the data is too big or it moves too fast or it exceeds current processing capacity* [10]. Big data has the potential to help companies to improve their operations and make faster and more intelligent decisions.

Despite all the aforementioned, big data is not that new. Although large data sets have been created in the last couple of years, big data has its roots in the scientific and medical communities, where the analysis of large amounts of data has been done for development, modeling and other forms of research, all of which processed large data sets [4], [7], [8], [10].

An example of big data might be petabytes of data consisting of billions of records, of thousands of relations, of millions of people, from various and different sources (e.g. web, sales, social media and mobile data).

To gain insights from this data, someone has to choose an alternative way to process it. Many approaches have been produced in order to deal with the volume, the velocity and the variety of massive data in which valuable information is hidden, which could not previously be seen because of the amount of work required to extract them.

In order to characterize the different aspects of big data, the three Vs of *volume*, *velocity* and *variety* are commonly used. They are used so to view and understand the nature of the data and the software platforms that are available, in order to exploit them.

As someone could understand, the power of big data is hidden in these three main features. The volume, the velocity and the variety, cooperate with each other, making the processing of big data more complicated, as the size of the data becomes larger, the speed of its growth is just a matter of seconds and it can contain thousands of different varieties of data [9], [11], [12], [13].

In Figure 4, we can see how these three characteristics interact with each other:

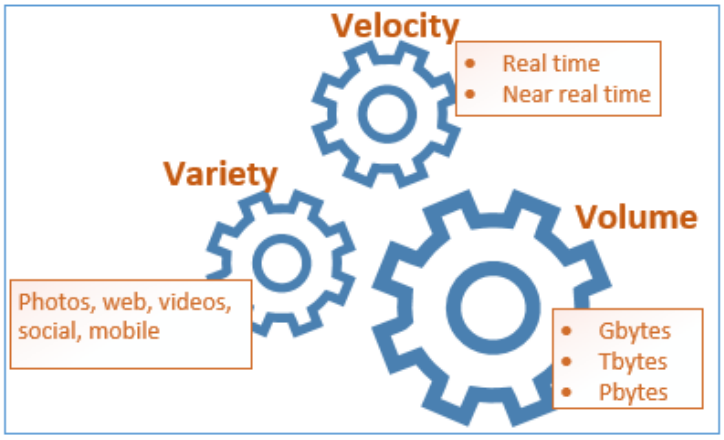

Figure 4 - Big data characteristics

- *Volume*: Big data is found in one size, large. Data can be found in the format of videos, music and large images. It is very common to petabytes of the storage system for enterprises. As the database grows, the applications and architecture built to support the data, has to be re-evaluated and re-constructed regularly. The benefit gained from the ability to process massive amounts of data is the main advantage of analyzing big data. By having more and more data, means that there could be created better models, including better results, and making better decisions. The big volume indeed, represents *big data* [9], [13].

- *Variety*: Big data can include data of all varieties and can be stored in multiple format. For instance, text, audio, video, click streams are log files are some of the aforementioned varieties. It is very uncommon that data presents itself in a form perfectly ordered and ready for processing. However, the organization has the need have meaningful information from them, so in order to do that it would be easier to have data in the same format. Unfortunately, it is not as easy as it looks, the most of the times. The real world have data in many different formats and that is the challenge we have to overcome with the *big data*. This variety of the data, represents *big data* [9], [12], [13].

- *Velocity*: The speed of the data growth have changed the way we look at the data, as there were times when we used to think that data of the previous month is recent and valid. Today, the data grows in real time so it needs to be analyzed quickly, taking in consideration that it is time sensitive. So this means that it must be processed every second and stored minute by minute into the archives of the enterprises, due to its growth speed. This high velocity data, represent *big data* [9], [11].

However, the complexity of big data does not end with just these three dimensions. Nowadays, there is a sum of technologies that are used to find the value that comes out of large data sets, by translating it to valuable information, making businesses to move forward.

Many of those technologies or concepts are not new, but nowadays they belong to big data [12], [13].
In short, these technologies include the following:

- *Business intelligence***.** It is a large category of applications and technologies built for gathering, storing, analyzing and providing access to various kinds of data. In general, *business intelligence is a set of techniques and tools for the transformation of raw data into meaningful and useful information for business analysis purposes* [14].
- *Data mining*. It is a process in which data is analyzed from various aspects and then transformed to valuable information. Its techniques focus on modeling and knowledge discovery for purposes of prediction, which helps in discovering new patterns from massive data sets.
- *Statistical applications*. Statistical applications are developed in order to deliver smaller amounts of data set, which can be used to study various data for estimating, testing and predictive analysis. Usually, the primary sources for gathering this kinds of information are surveys and experimental reports.
- *Predictive analysis*. It is a sub category of statistical applications in which the data is examined in order to result with predictions, based on information which was gathered from databases. Predictive analysis has as a main goal to identify the risks and the opportunities for the businesses, in which it is performed.
- *Data modeling***.** Data modeling is an application of analytics in which cases that answer to the question of "what-if" are usually applied via multiple algorithms to different kinds of data sets. In general, it is a process used to define and analyze data requirements needed to support the business processes.

The aforementioned categories are only an example of where big data is used and why it has a so large value for the businesses nowadays. That value is helping and encouraging the organizations to start using large repositories for storing data, in order to gain insights, retrieve information, reveal trends and uncover statistics to help them decide about their future movements. The last part, has helped the concept of big data to gain popularity, in combination with its associated tools, techniques, platforms, technologies and the various forms of analytics.

2.4 Knowledge discovery in databases (KDD)

The need for information has driven to the development of systems that can generate and gather large amounts of data. Many different kinds of businesses are participating in the field of gaining valuable outcomes. Some examples may include finance, banking, manufacturing, monitoring, health care and marketing.

For that reason, there are several tools currently being used in order to gain these useful and valuable insights. The information retrieval process, which is making use of the aforementioned tools is known as knowledge discovery in databases (KDD).

The basic task of the KDD process is to extract knowledge from lower level data (e.g. databases). In short, *knowledge discovery in databases is the process of identifying valid, useful and understandable patterns in data, for future use* [18]. In other words, the KDD process involves using the data stored in a database, and applying data mining methods to gain insights and information out of it. The main goal of the KDD process is to discover for unprocessed data, information or trends that may be useful. Extraction of knowledge from raw and unprocessed data is done by applying data mining methods, which is a step of the aforementioned process. KDD however, has a much broader scope.

The KDD deals with the process of discovering useful knowledge from data while data mining, statistical analysis and other techniques deal with only a particular step in this process. The KDD process tries to understand, to translate and to implement this information, for future cases and data sets.

The KDD is repetitive (one might have to move back to the previous steps), interactive and consists of various steps. The process starts with determining the KDD goals, and ends with the implementation of the discovered knowledge. Then the loop is closed. The retrieved knowledge is measured on the data sets, and the process starts for one more time. It is important to note that KDD is not accomplished without human interaction [15], [16], [17], [20].

The following steps, is a brief description of the nine (9) stepped KDD process, which starts with the managerial step. These steps can be seen in Figure 5 [15], [19], [20], [21].

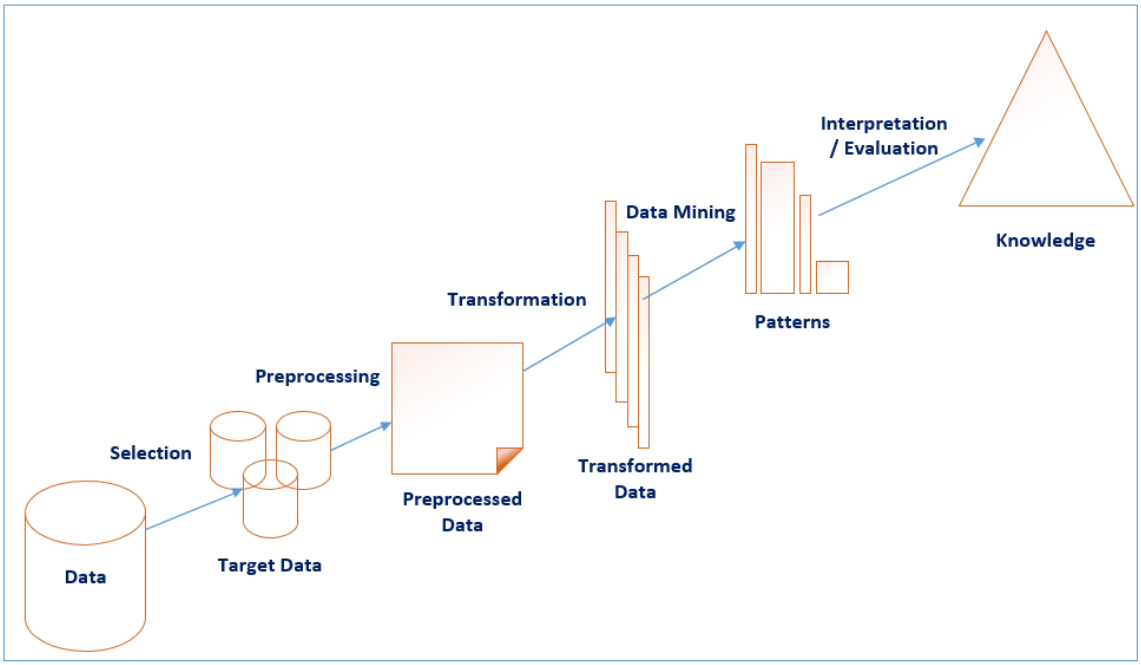

Figure 5 - Knowledge discovery in databases process

Step 1: The first step has to do with the understanding of what has to be done with the transformation, the algorithms and the representation. The people who are in charge of a KDD project have to understand and define the goals and the environment in which the knowledge discovery process will take place. After that, the pre-processing of the data starts. One should keep in mind that some of the pre-processing methods are similar to data mining algorithms, in order not to have any misunderstandings.

Step 2: The second step has to do with the creation of a data set among the whole data, where the process will be performed, according to the pre-defined goals. Some questions need to be answered such as what data is available, how can someone have more data and how the whole data can be integrated into one data set.

Step 3: The third step has to do with the data pre-processing and the data cleaning. The data is cleaned and is preprocessed, including removing noisy data, deleting or correcting errors, collecting information and deciding on strategies for manipulating missing data. This step, usually uses statistical methods or data mining algorithms.

Step 4: The fourth step is the step of data transformation where the generation of better data, is taking part. Such transformation methods are dimension reduction and extraction, attribute transformation and functional transformation. Furthermore, this step helps in finding useful features to represent the data depending on the final goals.

Step 5: The fifth step has to do with choosing the appropriate data mining task among classification, regression and clustering. This usually depends on the KDD goals which were set during the beginning of the process and also on the previous steps.

Step 6: The sixth step is about choosing the data mining algorithm, a method for searching and investigating various kinds of data patterns. In addition, this step includes deciding of which models and parameters might be appropriate according to the overall criteria and goal of the KDD process.

Step 7: During the seventh step, the chosen data mining algorithm is implemented. However, one should have in mind that the algorithm is deployed several times until a satisfying and valuable result is gathered. For instance, someone could deploy the kMeans algorithm various times, by changing the initial number of clusters of the kMeans Algorithm, until the best result will appear.

Step 8: As for the eighth step, the data mining results are evaluated and interpreted according to the goals that were defined in the first steps. It should be mentioned that in this step, the discovered knowledge is also written in documents for future usage, in case of similar problems or data sets, and that there can also be involved data visualization of the extracted information and models.

Step 9: During the final step, the discovered knowledge is being used and applied. The knowledge is transferred into different systems for further actions or is simply written and reported to the interested parties. The success of this step determines the effectiveness and the final outcome of the entire KDD process that was described above.

Having defined the basic steps and having introduced the KDD process, we now focus on the data mining component (step 5 of the KDD process), which has received the most attention at this thesis.

2.5 Data mining

Data mining can be used for prediction and description purposes, the fifth (5^{th}) step in the KDD process, mentioned in *section 2.4*.

Data Mining has to do with finding useful patterns and information among the data. However, there is a wide variety of definitions and criteria for performing data mining. In general, data mining can also be called as knowledge discovery, machine learning or predictive analytics. Although, it should be considered that each of these terms have a different meaning depending upon the context.

Data mining starts with data, which can range from a simple array of a few observations to a complex matrix of millions of observations with thousands of variables and relationships. The act of data mining uses specialized computational methods in order to discover meaningful and useful structures in the data. These *computational methods* have been taken and used from the field of statistics, artificial intelligence, machine learning, database theories, and pattern recognition.

As mentioned before, the first goal of data mining is to extract and generalize *meaningful patterns* from the data set. Its main objective is to find useful conclusions and information that can be used by the users of the analysis.

Algorithms used in data mining, are originated from the aforementioned computational methods, but in addition they are making use of several techniques such as parallel or evolutionary computing. Data mining is a non-stop process in which the data analyst is able to gain more information about the patterns and the relationships from data, during each data mining loop.

However in order for the data mining process to have results, various iterative *algorithms* are implemented, so to transform inputs to outputs. The application of algorithms for extracting useful outputs from the raw data, is the main difference between data mining and the other traditional data analysis techniques. Most of these algorithms were developed in the recent decades and are using ideas from the fields of machine learning and artificial intelligence. However, there are some algorithms that are based on the foundations of theories that are originated many years ago.

According to the data problem, data mining is classified into tasks such as classification, association analysis, clustering and regression, while each data mining task, uses specific algorithms like decision trees, k-nearest neighbors and k-means clustering, to finally extract and generalize meaningful patterns from the data set [22], [24].

2.5.1 Data mining types

There are various data mining types available, which have as a goal to extract knowledge from databases or data sets. These types, could be split into two main categories: "descriptive" and "predictive", as the primary goals of data mining in general, is to have results for *prediction* and *description* [22], [23], [24].

- As for the *prediction*, it has to do with using fields in a database, in order to forecast unknown or future values of other fields. *To be more specific, prediction refers to the creation of models that are capable of producing prediction results when applied to unseen, future cases* [25]. *Classification* and *regression* are the most frequent types of tasks that are applied in predictive data mining.

- As for the *description*, it focuses on finding human-interpretable patterns which describe the data. To be more specific, the description has to do with models that summarize and explain data for the purpose of inference. *Summarization* and *visualization* of databases are the main applications of descriptive data mining. The important fact of this concept is that it makes someone to see the data set from various levels of abstraction, which eases the examination of the general behavior of the data, since it is very difficult to derive that from a large database.

However, some of the predictive models can be descriptive, until the degree that they are understandable, and vice versa, but the distinction among them is useful for understanding the final goal.

Having understood the two major data mining types, it is time the different types of data mining learning models that are commonly used to solve the different data mining problems, to be mentioned. These are categorized into two (2) basic categories: the *supervised* and the *unsupervised* learning models [22], [23].

- *Supervised* data mining creates relationships based on labeled (known) training data and uses this created relationship, in order to map new unlabeled data. In addition, this kind of learning model can forecast the value of the output variables, according to a set of input variables, by developing a model from a training data set where the values of input and output are previously known.

- *Unsupervised* data mining finds hidden patterns in unlabeled (unknown) data, for which there is no previous knowledge. In unsupervised data mining, there are no output variables to predict, so its main objective is to find different patterns in data, according to the relationships among the points of the data set.

Having in mind all the aforementioned, the last goal of this section is to categorize the different data mining techniques, according to the data set we have and the results that

we want to derive from the process. These data mining techniques can be grouped into the following basic categories [24]:

- *Classification* has to do with the mapping of the data into one of several predefined classes. It is a data mining function that assigns the data points of a date set in a collection, to target categories or classes. The main goal of classification is to predict the aforementioned target category or class, while its tasks begin with a data set in which the data are labeled and previously known.

- *Regression* techniques are used in order to predict a target variable based on the input variables. It is a learning technique which maps a data point to a real-valued variable, while it is based on a generalized model built from a previously known data set. One should have in mind that always, the output variable is numeric.

- *Clustering* is the process of identifying natural groupings among the data set. It is a technique that assigns a data point into one of several groups (clusters). In the case of clustering, the groups must be determined from the data, unlike classification in which the groups are already defined. Generally, clusters are defined by discovering natural groupings of data items based on similarity metrics, probability density models or the distances between the data.

- *Anomaly detection* is able to identify the data points that are significantly different from the other data points among the data set. One of the most prolific applications of anomaly detection, is credit card transaction fraud detection.

- *Text mining* is a data mining application where the input data is text, which can be found in the form of documents, messages, emails and web pages. To help the process of data mining among text data, the text files are converted into document vectors, where each unique word is considered as a different attribute. Once the text file is converted to document vectors, standard data mining tasks such as classification or clustering, can be applied on these document vectors.

2.5.2 Data mining algorithms

An algorithm, as we have already seen in the previous chapters, is a step-by-step procedure for solving a problem. In data mining, it is the way with which a particular data problem is solved.

Many of the algorithms have a number of different steps which are repeated various times until a limiting condition is met or until the desire result outcomes.

A data mining classification task can be solved using various approaches and algorithms such as decision trees, k-nearest neighbors (k-NN), Bayesian Models, density based models and regression algorithms.

In general, the choice of which algorithm to use according to the problem that has to be solved, depends on the following:

- Type of data set
- Objective of the data mining
- Structure of the data
- Computational power
- Number of records and attributes
- Available time

The analyst has to decide about what algorithm has to use, by evaluating and testing the performance of multiple algorithms.

There have been hundreds of algorithms developed in the last few decades to solve the various data mining problems that may appear, especially now that we have to deal with the big data phenomenon [23].

At this thesis, as it was mentioned on *Chapter 1*, the main goal was to implement a data mining algorithm to run in parallel on top of an OLAP database. For that reason, the data mining algorithm that was chosen to be implemented was the fuzzy kMeans algorithm. The aforementioned algorithm belongs to the category of clustering algorithms, and it is mostly used for unknown and unlabeled data, in order to assign data points to multiple clusters with different degree of association to each one.

As a result of this, we will not mention any further details for the other data mining tasks, apart from the task of *clustering*, which will be our main interest during the thesis. Answers will be given to questions such as how the clustering techniques are performed, when should we prefer the clustering algorithms and why in general clustering can help us to gain information out of different kinds of data.

Further details about how the fuzzy kMeans Clustering Algorithm works and clusters data, are going to be given in chapter 5 (chapter of the algorithm implementation).

Table 1, provides a summary of some data mining tasks with the most commonly used algorithms, followed by some examples.

Tasks	Algorithms	Examples
Classification	▪ Decision trees ▪ Neural networks ▪ Bayesian models ▪ k-nearest Neighbors	Assigning new customers into one of the known customer groups, according to their behavior
Regression	▪ Linear regression ▪ Logistic regression	Predicting unemployment rate for next year
Anomaly Detection	▪ Distance based ▪ Density based ▪ Local Outlier Factor (LOF)	Fraud detection in credit cards
Clustering	▪ kMeans ▪ fuzzy kMeans ▪ Density Based clustering (DBSCAN)	Partition customers in groups, according to their similarities

Table 1 - Data mining tasks

2.6 Clustering

Clustering can be considered as the most important *unsupervised learning* technique, *where a set of patterns in a multi-dimensional space, are grouped into clusters in such a way that patterns in the same cluster are similar in some sense and patterns in different clusters are dissimilar in the same sense* [37]. A cluster, in general, is a collection (group) of objects which are similar between them and are dissimilar to the objects belonging to other clusters.

We can show this with a simple graphical example (Figure 6):

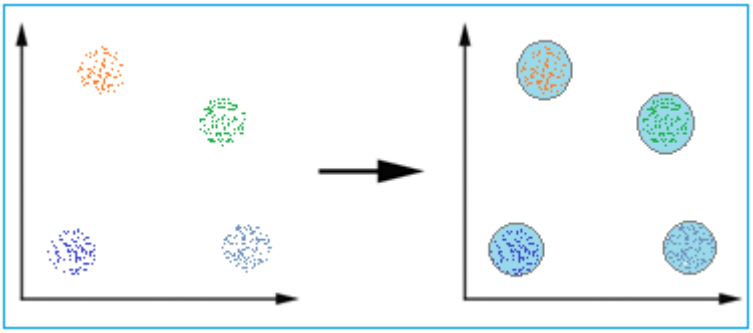

Figure 6 - Clustering data

Each circle represents one cluster containing several points. In this simple example (Figure 6), the circles obviously represent the best clustering of the data points into four clusters based on the distance of the points, to the cluster centroids. Circles are a good way to think of clusters, because clusters are also defined by a center point and radius [26], [28], [33], [34].

The center of the circle is called the centroid (mean) of that cluster. It is the point whose coordinates are the average of the coordinates of all the data points in the cluster.

In this case we can easily identify the four (4) clusters into which the data can be split. It should be mentioned that for the clusters above, the similarity criterion is the distance. In general, it must be said that two or more points belong to the same cluster if they are close according to a given distance. As a result, this type of clustering is called *distance-based clustering*. Some of the most used distance measures are the following [28], [35]:

- Euclidean Distance Measure
- Squared Euclidean Distance Measure
- Manhattan Distance Measure
- Cosine Distance Measure
- Tanimoto Distance Measure
- Weighted Distance Measure

Another kind of clustering is *conceptual clustering*, where two or more data points are grouped according to their fit to a given description, not according to simple similarity measures, like the previous kind of clustering.

Depending on the clustering technique used, the number of groups or clusters is either user defined or automatically determined by the algorithm. *Its main objective is not to predict a target class variable, but to simply capture the possible natural groupings in the data* [26].

2.6.1 Clustering applications

Clustering has a wide variety of applications, ranging from market partition to customer partition, and web analytics [26], [29].

The most common application of clustering is to explore the data and find all the possible meaningful groups that the data can be split. Some of the most common applications of clustering are:

1. *Marketing*: Clustering can be used to find the common groups of customers based on all the past customers' behaviors, attributes or purchase history.

2. *Document clustering*: It automatically groups the text documents into groups of similar topics. It provides a way of identifying key topics, understanding and summarizing these clustered groups, rather than reading through the whole documents which is a time consuming process.

3. *Session grouping*: In web analytics, clustering can be helpful to understand clusters of *clickstream patterns* and discover different kinds of *clickstream profiles*.

4. *Reduce Dimensionality*: In an n-dimensional data set, the attributes can be reduced to less categorical attributes. So, the complexity is reduced, although there is loss of valuable information because of the dimensionality reduction to less attributes.

5. *Object Reduction*: It can be used to find the most common representation of all the data points that belong to the same cluster and it can be a new point whose values are the means of the values of all the points that belong to the cluster.

6. *Search Engines*: Clustering algorithms are playing a significant role behind the search engines. Search engines try to group similar objects in one certain cluster, while they move dissimilar objects far from each other. So, clustering provides results for the data that is searched, according to the nearest similar or identical object, which belong to the cluster around the searched data.

2.6.2 Clustering techniques

Clustering techniques can be classified based on the algorithmic approach which is used in order to find clusters in the data set [30], [31], [32].

- *Partitioning clustering*: In a data set of *n* objects, a partitioning method develops *k* partitions of the data, where each partition represents a cluster ($k \leq n$). Partitioning clustering divides the data into *k* groups, such that each group must contain at least one object. Most of the partitioning methods typically obey to the rule that each object must belong to exactly one cluster. The general idea of a good clustering is that objects which belong to the same cluster are close to each other, whereas objects in different clusters are far apart.

- *Density-Based clustering*: Its general idea is to continue growing a specific cluster as long as the density (number of objects or data points) in the "neighborhood" exceeds some given threshold. A cluster can be defined as a region where data points are gathered, surrounded by a low-density area where data points are sparse. Each area with high density can be assigned to some cluster and the low-density area can be considered as faulty or noisy data points. However, in this form of clustering not all data objects are clustered since noisy data points are discarded and not assigned to any clusters.

- *Hierarchical clustering*: Hierarchical clustering is a process where a cluster hierarchy is created based on the distance between the various data points. The output is a tree diagram that *shows the different clusters at any point of precision which is specified by the user* [33]. There are two approaches to create an hierarchy of clusters:

    - A *bottom-up approach* (agglomerative) starts with each data point forming a separate cluster. Afterwards, it merges the data groups close to one another, until all the groups are merged.
    - The *top-down approach* (divisive) starts with all the data points belonging to the same cluster. Afterwards, in each iteration, a cluster is divided into smaller sub-clusters, until each data point gets to one cluster.

- *Model (Distribution)-Based clustering*: In Model-Based clustering, *a cluster can be thought of as a grouping that has the data points belonging to the same probability distribution* [35]. So, each cluster can be represented by a distribution model (like Gaussian or Poisson), where the parameter of the distribution can be iteratively optimized between the cluster data and the model.

In Table 2, the main characteristics of each clustering method are described, according to their type:

Method	General Characteristics
Partitioning clustering	▪ Finds exclusive spherical clusters ▪ Is distance based ▪ May use an average number to represent the cluster centroid ▪ Effective for small-medium sized data sets.
Hierarchical clustering	▪ Clustering is a hierarchical decomposition ▪ Cannot correct faulty merges or splits ▪ May uses techniques like micro-clustering
Density-Based clustering	▪ Finds clusters of various shapes ▪ Can handle noisy data ▪ Each point must have a minimum number of points within its region ▪ Needs density parameters as termination condition
Model (Distribution) - Based clustering	▪ Data is generated by a mixture of underlying probability distribution ▪ Optimizes the fit between the data and mathematical models

Table 2 - Clustering method's characteristics

2.6.3 Clustering algorithms

Clustering algorithms can be categorized based on their cluster model, as listed in *section 2.6.2*. In general, there are hundreds of published clustering algorithms, where not all provide models for their clusters and as a reason for that they cannot easily be categorized and classified.

Some clustering algorithms use the basic concepts and ideas of multiple clustering methods, and for that reason it is complicated most of the times to characterize that a given algorithm belongs to only one clustering method. Moreover, there are some applications that may have clustering criteria which require the integration of several clustering techniques, at the same time, in order to output the final results.

For all the aforementioned, it can be assumed that there is not an objectively "correct" clustering algorithm, but one should have in mind that for a particular problem, the most appropriate clustering algorithm has to be chosen by testing and experiments, unless there is a specific reason to prefer a certain clustering model instead of a different one [26], [34], [35].

Some of the most used and well known clustering algorithms, are listed in Table 3:

Method	General Characteristics
Partitioning Clustering	▪ Fuzzy kMeans ▪ kMeans ▪ CLARA ▪ CLARANS
Hierarchical Clustering	▪ CURE ▪ CHAMELEON ▪ BIRCH ▪ SLINK
Density-based Clustering	▪ DBSCAN ▪ OPTICS ▪ DENCLUE ▪ CLIQUE
Model (Distribution)-Based Clustering	▪ Expectation Maximization ▪ COBWEB ▪ CLASSIT ▪ Self-Organizing Feature Map (SOM)

Table 3 - Clustering algorithms

After having studied in deep details the chapter of the state of the art on data analysis, the next goal is to study about the distributed processing framework of Hadoop, which offers distributed computational and storing techniques, to each project that is making use of it. As we have to implement our own distributed way of computing and storing the various clustering results of the fuzzy kMeans clustering algorithm, we have first to understand how the whole implementation of Hadoop functions in general.

Chapter 3

Apache Hadoop

Summary: *"In this chapter, a deep-detailed study is done about what is the apache Hadoop. During the research, the components and the ecosystem of Hadoop are stated in a detailed way. Finally, the MapReduce paradigm and the Hadoop's distributed file system are explained, which we are going to fully replace in the chapter of the implementation, after having discussed about them in details."*

3.1 Apache Hadoop: A distributed processing framework

Today, we are surrounded by data. People upload videos, take pictures on their mobile phones, send messages to friends, update their social network profiles, make comments around the web or click on advertisements. As a result of that, machines have to generate and store massive amounts of data, to provide it back to their owners and to gain valuable information out of it.

This data growth, created multiple difficulties to the businesses of all over the world such as Google, Yahoo and Amazon. The aforementioned businesses, had every day to deal with millions of bytes of data in order to discover which websites were mostly visited, what books were in demand and what type of advertisements were more amusing to the people. However, all the existing tools were becoming incapable to process and derive results from such large amounts of data. For that reason, Google published the MapReduce framework, a different system used to scale their big data processing needs.

Around 2004, Google published two (2) papers describing the Google file system and the MapReduce framework. Google adopted these two technologies, in order to scale and help the complex work of its own search system. A new project was produced in order to combine better these two technologies, and as a result of that, *Hadoop* was born [7], [38], [40], [45].

3.1.1 What is Hadoop?

Today, we live in the age of big data, where the data volumes we need to work with, demand more and more storage and processing capabilities. Big data come with two initial challenges:

- How to store and work with extremely large data
- How to understand data and turn it into advantage.

Hadoop helps the market by storing and providing computational capabilities over massive amounts of data. *It provides and supports the development of open source software that supplies a framework for the development of highly scalable distributed computing applications.* [39] Furthermore, the Hadoop framework deals with the processing details, leaving the developers free to focus on the development of applications logic.

In a few words, Hadoop is a *distributed system* made up of a *distributed file system* and it *offers a way to parallelize and execute programs on a cluster of machines*.

In Figure 7, we are able to see how one interacts with a Hadoop cluster. In general, a Hadoop cluster is a set of commodity machines which are networked together in a single location. Data storage and processing take place within this region of machines, while different users are able to submit jobs which require computational processes to Hadoop, using different machines (clients) in remote locations from the Hadoop cluster [40], [41], [42], [48].

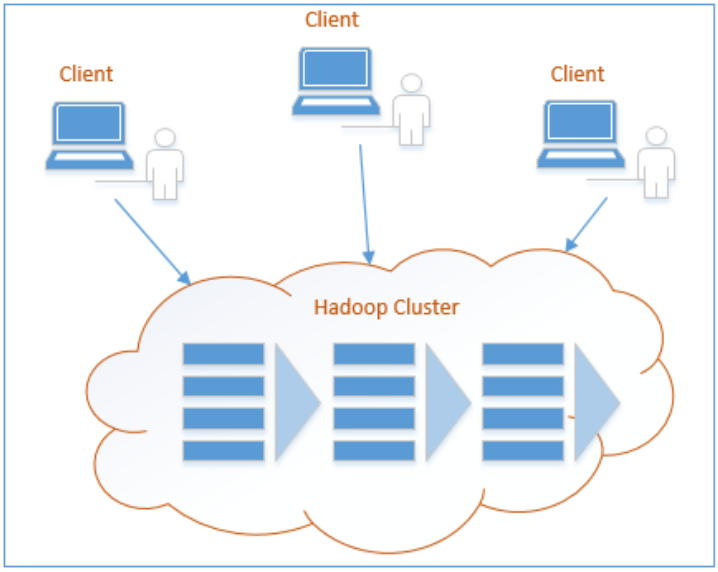

Figure 7 -Hadoop cluster

In other words, Hadoop is an open source framework for writing and running distributed applications that process and store massive amounts of data and can run simultaneously. In Figure 8, we can see the two main components of Hadoop that help into the idea of distributed computation and storage (they will be described later). Hadoop, changes the way that data is generally managed and processed, *by leveraging the power of computing resources composed of commodity hardware, while it can automatically recover from failures* [50].

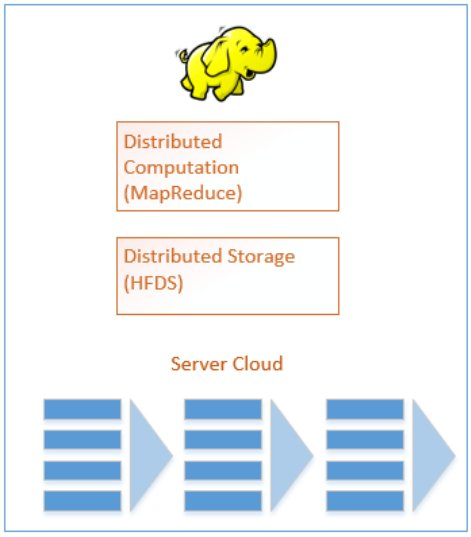

Figure 8 - Hadoop's components

The differences of Hadoop as for the distributed computing in general, that makes it so different and special, are listed below. In general, as for its computational techniques, Hadoop offers:

- *Accessibility*: Hadoop runs on large clusters of machines, or on cloud computing services, making it easily accessible by every kind of client.
- *Robustness*: As Hadoop is intended to run on commodity hardware, it is designed with the idea of having various hardware malfunctions, so it can manipulate failures in a better way.
- *Scalability***:** Hadoop scales and can be configured, in order to be able to handle larger data by adding multiple nodes to the cluster.
- *Simplicity*: Hadoop allows users to quickly write efficient and effective parallel code, making it even easier to test it, by using its distributed computational techniques and storage.

Hadoop, as shown in Figure 9, is generally a distributed master-slave architecture that consists of the Hadoop distributed file system (HDFS) for storage and the MapReduce framework for computing [38], [42].

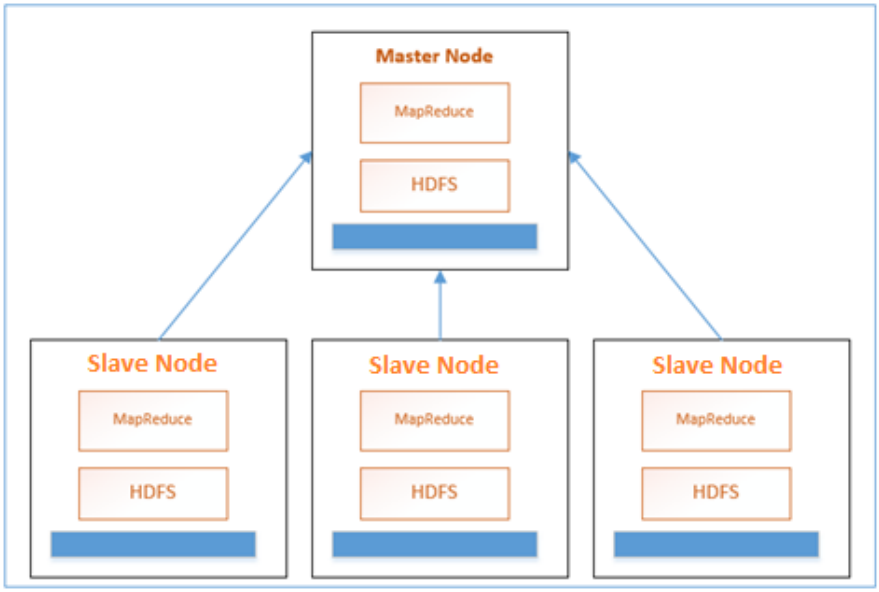

Figure 9 - Master-slave architecture of Hadoop

HDFS is the storage component of Hadoop. It is a distributed file system which is developed for having high throughput and works best when it interacts with large files. In order to support this throughput, *HDFS creates multiple block sizes and data locality optimizations, so to reduce network input/output* [40]. HDFS, also offers scalability and availability, which are achieved because data is split into smaller parts and HDFS can manipulate various types of errors. In short, *HDFS replicates files for a configured number of times, it is tolerant of both software and hardware failure and automatically re-replicates data blocks on nodes that have failed* [51]. In Figure 10, we can see a logical representation of the components in HDFS, the NameNode and the DataNode, which will be described later. It also shows an application that is using the Hadoop file system library, in order to access the HDFS.

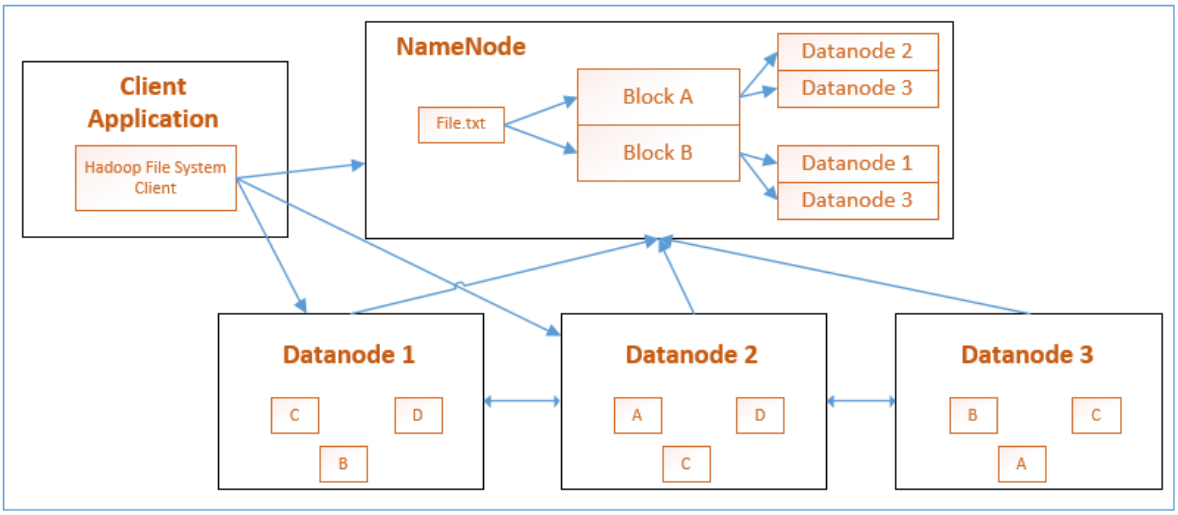

Figure 10 - HDFS components

MapReduce is a distributed computing framework which allows the parallelization of the work over a large amount of raw data. This type of work, which could take a long time to finish using sequential programming techniques, can be completed in a few minutes using MapReduce, by working on a Hadoop cluster. The MapReduce abstracts away the complexities which someone faces with when working with distributed systems (e.g. computational parallelization, work distribution). In that way, MapReduce allows the programmer to focus on the application development, by splitting work submitted by a client, into small parallelized map - reduce workers (Figure 11).

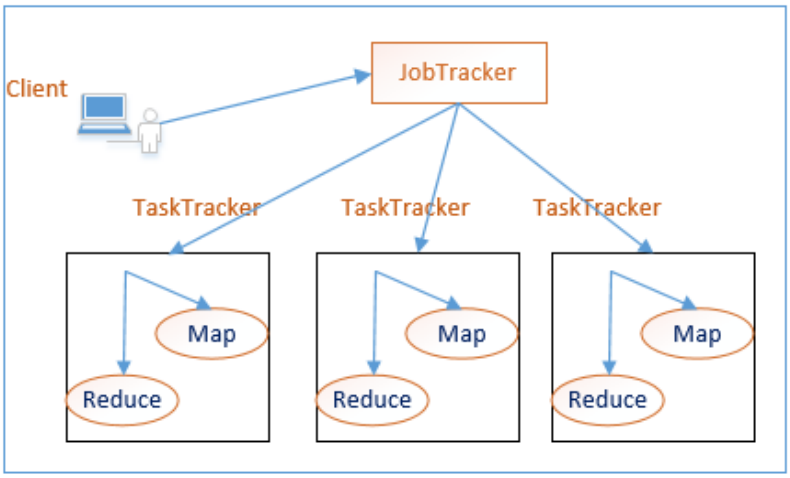

Figure 11 - MapReduce paradigm

In general, MapReduce provides a model which transforms complex computations into computations over a set of <key, value> pairs. It schedules and monitors the MapReduce jobs and is responsible for executing again the failed tasks.

So, the role of the programmer becomes much easier, as he has only to define the map and the reduce functions, where the map function outputs <key, value> pairs which are then processed by the reduce functions according to the keys of these pairs, in order to produce the final output.

More details about how HDFS and MapReduce work, will be mentioned in the following chapters of the thesis [41], [42], [45], [46], [47].

3.1.2 Hadoop cluster components

A typical Hadoop environment generally consists of a MasterNode along with multiple SlaveNodes (Figure 12). Each of these nodes consist of several specialized software components, which we will described next [38], [41], [42], [48].

As we can see in Figure 12, the *HDFS layer* consists of two different types of nodes, the *NameNode* and the *DataNode*, while the *MapReduce layer* consists of two types of trackers, the *JobTracker* and the *TaskTracker*.

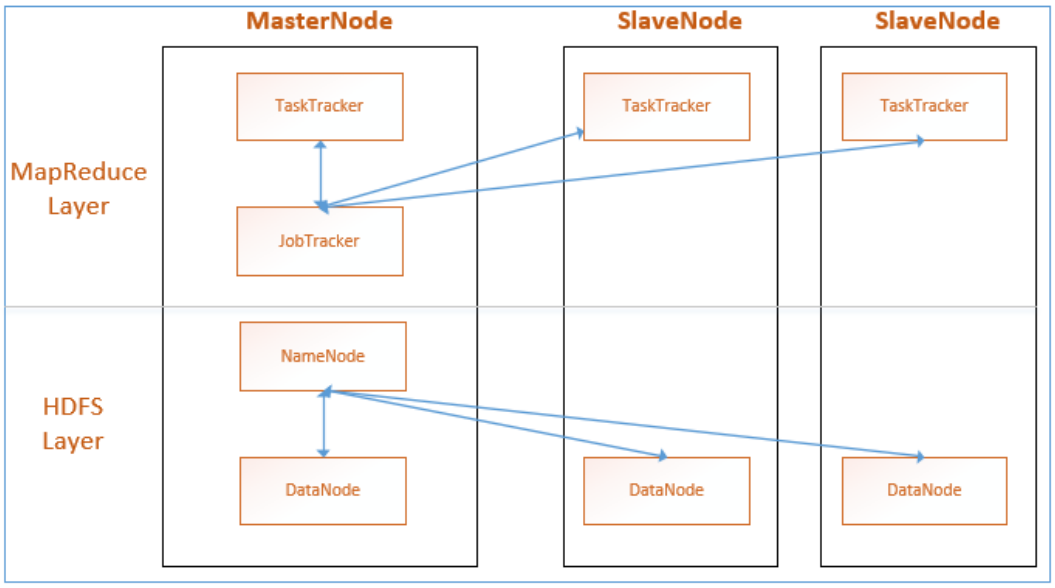

Figure 12 - Hadoop cluster components

In general, most of the Hadoop deployments consist of more than one MasterNode instances, and for that reason, the risk of a single point of failure is eliminated.

The major elements that are presented in the *MasterNode* are:

- *JobTracker*: It is assigned to interact with client applications. It is also responsible for distributing MapReduce tasks to particular nodes within a cluster. The JobTracker daemon combines the application with Hadoop. By the time that the code is submitted to the cluster, the JobTracker determines the files that have to be processed, assigns the nodes to different tasks and monitors all tasks while they are running. In case that one of these tasks fail, the JobTracker re-executes the task, either on the same or on a different node. It has to be mentioned that there is only one JobTracker daemon for each Hadoop cluster.

- *TaskTracker*: It is a process in the cluster that receives tasks from the JobTracker. The TaskTrackers deal with the execution of individual tasks on each slave node. Usually, there is a single TaskTracker for each slave node, but each TaskTracker can *spawn multiple JVMs to handle many tasks at the same time* [47]. The TaskTracker has to communicate with the JobTracker, so in case that the JobTracker fails to receive a *heartbeat* from a TaskTracker within a short amount of time, it will be assumed that the TaskTracker has failed and will submit again the failed tasks to different nodes of the cluster.

- *NameNode*: The NameNode is the *bookkeeper* of HDFS, as it writes down the way that the files are split into file blocks, the nodes that store these blocks and the overall state of the DFS. The Client applications contact with NameNodes when they need to find or edit a file. The NameNode is the master of HDFS, which is responsible for directing the DataNode daemons. The server which hosts the NameNode typically does not store any data or perform computations, in order to shrink the workload of the machine.

- *DataNode*: The DataNode has to store data in the HDFS and is responsible for replicating data across clusters. When someone wants to read or write an HDFS file, the file is split into blocks and the NameNode informs the client about which DataNode each block resides in. For that reason, the client speaks directly with the DataNode daemons in case that he wants to edit the files corresponding to the blocks. Furthermore, a DataNode interacts with other DataNodes in order to replicate its data blocks for redundancy and have to make reports to the NameNode. The first job of the DataNode is to inform the NameNode of the blocks that is currently storing, which gives instructions such as to create, move, or delete blocks from the local disk.

On the other hand, apart from the MasterNode, there are also the SlaveNodes which unlike the MasterNode, a Hadoop deployment consists of hundreds of SlaveNodes, which have capabilities and power to analyze terabytes of data. Each SlaveNode includes a *DataNode* and a *TaskTracker* [42], [45], [49].

3.1.3 Hadoop architecture

The architecture of a Hadoop-based system can be described with Figure 13:

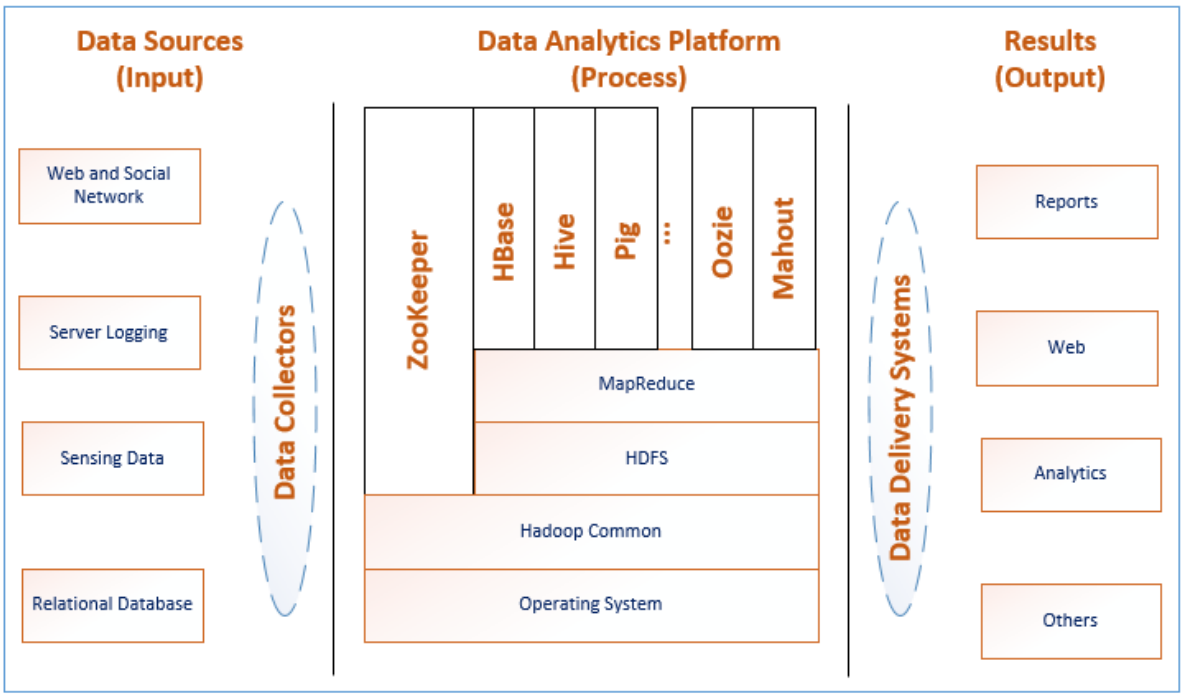

Figure 13 - Hadoop architecture

The Hadoop ecosystem grows by the day. It is impossible to keep track of all of the projects that interact with Hadoop in some form, and are using it in order to have distributed storage and computational techniques.

A typical Hadoop-based platform includes the Hadoop distributed file system (HDFS), the parallel computing framework (MapReduce), a column-oriented data storage table (e.g. HBase), high-level data management systems (e.g. Pig and Hive), a big data analytics library (e.g. Mahout), a distributed coordination system (e.g. ZooKeeper), a workflow management module (e.g. Oozie), data transfer modules (e.g. Sqoop), data aggregation modules (e.g. Flume), and data serialization modules (e.g. Avro) [41], [46], [48].

As we can see in Figure 14, some of the projects that consist the Apache Hadoop's Ecosystem, are the following [38], [49]:

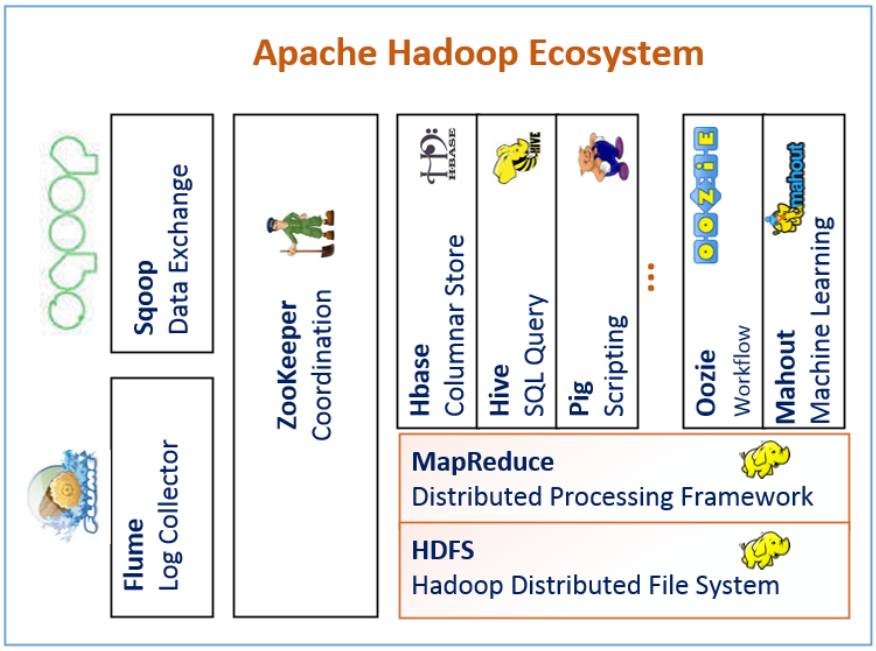

Figure 14 - Apache Hadoop ecosystem

Apache Pig: Pig is a platform for analyzing large data sets that consists of a high-level language, in order to express data analysis programs, in combination with infrastructures for evaluating these programs. What makes Pig so special is that its structure is *amenable to substantial parallelization* [52], which in aids it handling very large data sets. Until now, Pig's infrastructure layer consists of a compiler that produces sequences of MapReduce programs. Pig's language layer currently consists of a textual language called Pig Latin, which is easy to use, optimized and extensible.

Apache Hive: Hive is a data warehouse system for Hadoop that helps with the data summarization, the queries and the analysis of large datasets that are stored in Hadoop. It provides a mechanism to query the data using a SQL-like language called HiveQL. It also allows traditional MapReduce programmers to plug in their own map and reduce jobs when it is hard on inefficient to express it in HiveQL. Like apache Pig, Hive's runtime engine translates HiveQL statements into a sequence of MapReduce jobs for execution.

Apache Oozie: Apache Oozie is a server-based workflow engine which is specialized in running workflow jobs with actions that run MapReduce and Pig jobs. It is data aware and coordinates jobs according to their dependencies. Furthermore, Oozie has been integrated with Hadoop and can support any kind of Hadoop jobs.

Apache HBase: Apache HBase is an open source, distributed, and column-oriented data store. It was built on top of Hadoop and HDFS for the underlying storage and supports computations using MapReduce. The main components of HBase, are the *HBase Master* who is responsible for negotiating load balancing across all region servers and maintain cluster's health and the *RegionServer* which is deployed on each machine and hosts data, processing I/O requests. It is scalable, meaning that it can host very large tables, containing billions of rows and millions of columns.

Apache Mahout: Apache Mahout is an open source scalable machine learning library based on Hadoop. It implements many different approaches to machine learning and currently contains implementations of algorithms for classification, clustering and collaborative filtering. Mahout is scalable, and as a result of that it scales to reasonably large data sets, by leveraging algorithm properties or implementing versions based on apache Hadoop.

Apache ZooKeeper: Apache ZooKeeper is a centralized coordination service for large scale distributed systems. It keeps and watches the configuration and the naming information, by providing distributed synchronization and group services for distributed systems applications. It has to be mentioned that ZooKeeper makes the HBase to be operational.

Apache Sqoop: Apache Sqoop is a tool for moving bulk data between apache Hadoop and structured data stores such as relational databases, by providing various command-line suites to transfer the data.

Apache Flume: Apache Flume is a tool for collecting log data in distributed systems. It consists of a flexible and fault tolerant architecture, while it is mostly used for collecting log data from many diverse sources and moving them to a specific data store.

Apache Avro: Apache Avro is a fast data serialization system for Hadoop which is making use of JSON, in order to define the data types and protocols, and to serialize data in a binary format. That serialized data is coupled with the data schema, which makes easier its processing with the use of various programming languages.

3.2 Apache Hadoop ecosystem

As it was mentioned earlier, Hadoop has a distributed master-slave architecture whose core components are the Hadoop distributed file system (HDFS), which is used for distributed storage and the MapReduce framework, which is used for distributed computational capabilities. In the following sections, there are given more information on how these two components work and communicate, aiding the goal of Hadoop.

3.2.1 Hadoop distributed file system (HDFS)

The Hadoop distributed file system (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with the existing distributed file systems, but the differences from them can be significant.

Generally speaking, Hadoop distributed file system (HDFS) is a fault-tolerant distributed file system that has been optimized for streaming reads on large files and is suitable for applications that have large data sets. In addition, HDFS uses a simple model for data consistency where files can only be written to once.

HDFS is making use of a concept called block replication to replicate data across the various nodes in the cluster. In general, HDFS uses a much larger block size when compared to desktop file systems. By the time that a file has been stored into the HDFS, it is split into one or more data blocks, while copies of these blocks are distributed to nodes in the cluster to have high data availability in the case of a disk failure. The number of copies that HDFS has to make for each data block is determined by the replication factor setting. By default, the replication factor is 3, meaning that three replicas of a data block will be distributed across the nodes in the cluster [53], [54], [55].

HDFS Architecture

As we have already discussed, HDFS has a master-slave architecture (Figure 15)
[56], [57].

A HDFS cluster consists of a *NameNode*, which is a master server that manages the file system and deals with the access to files by clients.
Furthermore, there are a number of *DataNodes*, usually one per node in the cluster, which deal with the storage attached to the nodes that they run on.

HDFS allows user data to be stored in files where a file is split into one or more blocks, while these blocks are stored in a set of DataNodes. The NameNode is able to open, close and rename files and directories, and it determines how the blocks will be mapped to the DataNodes. The DataNodes server read and write requests from its clients and

can also perform block creation, deletion, and replication if they are told to by the NameNode.

In addition, apart from these, the HDFS includes a secondary NameNode, which make some people think that when the primary NameNode goes offline, *the secondary NameNode takes over. In fact, the secondary NameNode connects with the primary NameNode and builds check-points of the primary NameNode directory information, which the system then saves to local or remote directories* [58]. These check-points can be used to restart a failed primary NameNode without having to re-execute the previous actions, and then to edit the log to create an up-to-date directory structure.

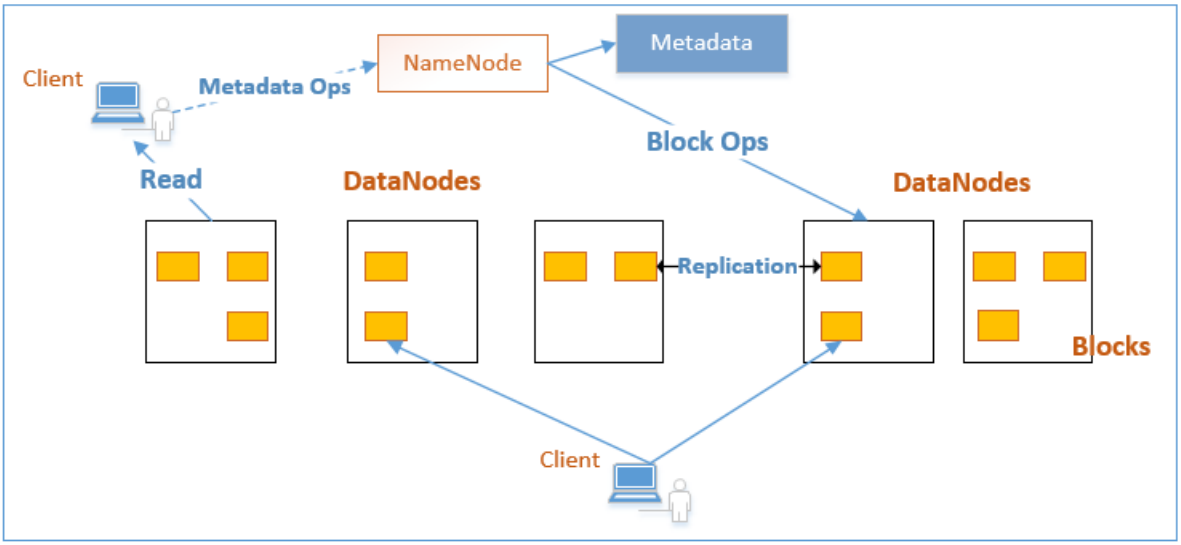

Figure 15 - HDFS architecture

The file content is split into blocks (typically 128 megabytes), while each block of the file is independently replicated at multiple DataNodes. The blocks are stored on the local file system on the DataNodes, while all blocks in a file, apart from the last block, have exactly the same size.

The NameNode periodically receives a *Heartbeat* and a *BlockReport* from each of the DataNodes in the cluster. The *Heartbeat* means that the DataNode is functioning well. In the case that a replication of a block is lost because of a failure, the NameNode creates another block replication. The NameNode sends instructions to the DataNodes by replying to heartbeats which include commands to: replicate blocks to other nodes, remove local block replicas, re-register and send an immediate block report, or shut down the node.

HDFS is built using the Java language, which means that HDFS can be deployed on a wide range of machines, and as a result of that any machine that supports Java can run the NameNode and the DataNode software [55], [57].

3.2.2 MapReduce framework

MapReduce is a relatively new technology, which builds upon much of the fundamental work from both mathematics and computer science. MapReduce concept is that of "divide and conquer", where a single problem is split into multiple individual sub-problems. However, this approach becomes even more desiring when the sub-problems are executed concurrently, at the same time.

Hadoop MapReduce is a software framework for easily writing applications which can process large amounts of data in-parallel, on large clusters of commodity hardware in a reliable and fault-tolerant way.

In this programming paradigm, *applications are divided into self-contained units of work* [64], which can be run on any node in the cluster. In a Hadoop cluster, a MapReduce program is known as a *job*, which is run by being broken down into smaller parts, known as *tasks*, which are programmed to run on the nodes in the cluster where the data exists [59], [60], [65], [66].

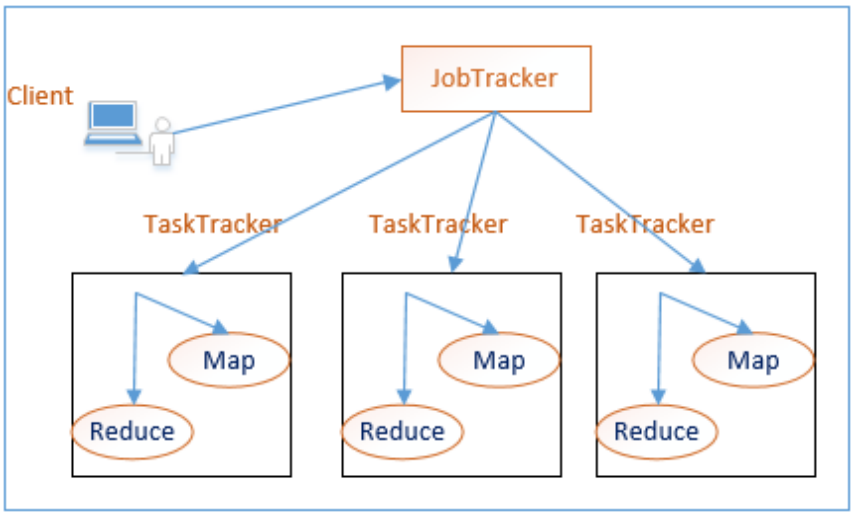

Figure 16 - MapReduce tasks

As we can see in Figure 16, the applications submit jobs to a specific node in a Hadoop cluster, which is running a JobTracker. The *JobTracker* is responsible to communicate with the NameNode to see if all of the data which is required for the job is available across the cluster.

The job is then broken into *map tasks* and *reduce tasks* for each node in the cluster to work on. The *JobTracker is trying to schedule tasks on the cluster where the data is stored, rather than sending data across the network to complete a task* [64].

However, the MapReduce framework and the HDFS typically exist on the same set of nodes, so the JobTracker can easily schedule tasks on nodes where the data is stored.

- The *map tasks*, take a set of data and transform it into another set of data, where individual elements are broken down into <key, value> pairs.
- The *reduce tasks*, take the output from a map task as an input and combine those pairs into a smaller set of pairs.

A set of programs that run continuously, known as *TaskTrackers*, watches the status of each map and reduce task. In the case that a task fails to complete, the failure is reported to the JobTracker, which reschedules the task on another node in the cluster.

The whole MapReduce concept, provides a great range of scalability, while it also maximizes parallelism by manipulating data stored across multiple clusters. The data sets for Hadoop MapReduce are stored in the HDFS in data blocks which can be processed independently by map tasks in parallel. MapReduce applications do not have to be written in Java, though most MapReduce programs that run natively under Hadoop are written in Java [61], [62].

As we can see in Figure 17, the MapReduce framework uses the data blocks which are stored in the *HDFS* as input to *map tasks*. The *JobTracker* is responsible for the distribution of the map and the reduce tasks to the TaskTrackers, running on each of the machines. All data are transformed in the form of <key, value> pairs. After the map tasks, the outputs are *shuffled* and sorted in the same group, based on their keys, so that all the values with the same key will be given to the same *reducer tasks*. The output of the reduce task is written back to the *HDFS* [60], [61], [62], [63].

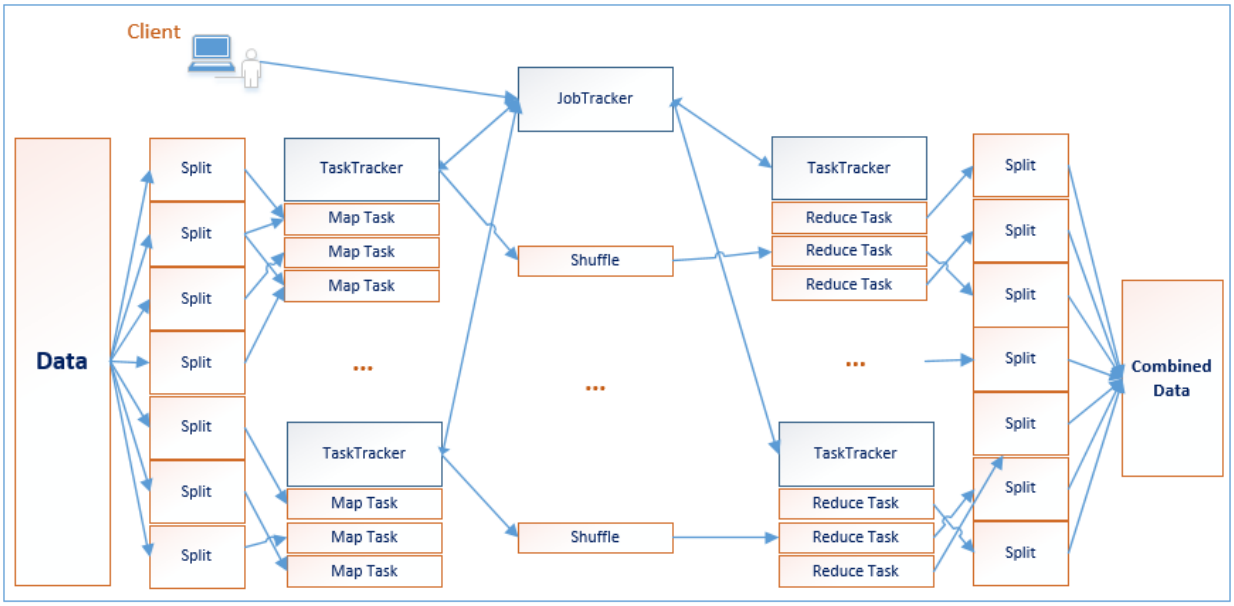

Figure 17 - MapReduce framework

MapReduce Example

In Figure 18, we can see a simple MapReduce example, which counts the words that are repeated in a small text file.

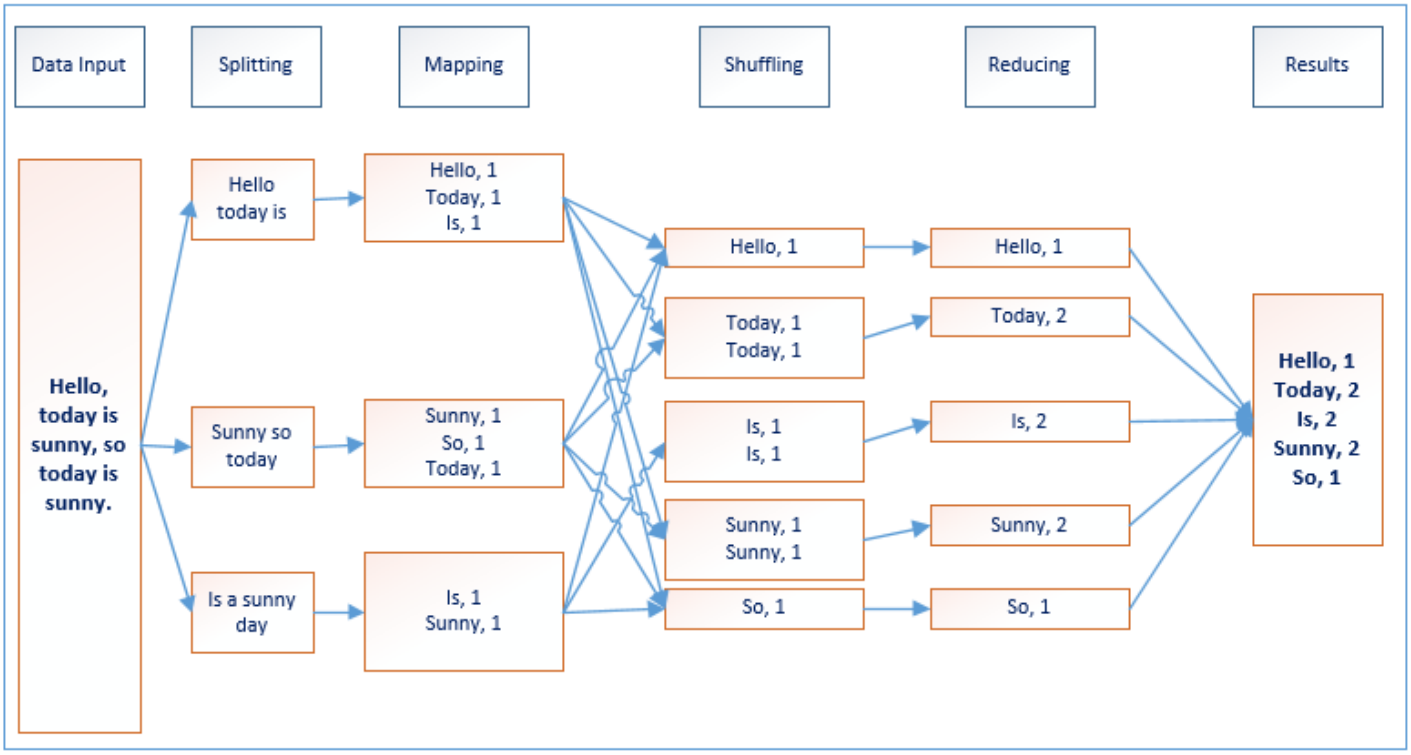

Figure 18 - MapReduce example

According to that example, we can easily define the three different phases that a MapReduce job is split:

- Firstly, we have the *pre-Map phase*, where the data is read from the file system and is partitioned to blocks.
- Then, during the *Map phase*, each map task reads the blocks, and transforms the data into <key, value> pairs.
- During the next phase, the *Shuffle phase*, the data with the same keys are grouped and sorted together, keeping their <key, value> pair format.
- Afterwards, the *Reduce phase* starts, where each group of the *Shuffle phase* is assigned to a single reduce task, which combines the values with the same keys.
- Finally, the *result* of the Reduce phase is written back to the file system, having an obvious smaller size than before.

Chapter 4

Apache Mahout

Summary: *"In this chapter, the data mining software of apache's Mahout is explained. Apache Mahout belongs to the Hadoop's ecosystem, and as our goal is to understand how it performs parallel clustering with massive data, we have to fully understand how the clustering techniques are fulfilled with Mahout. For that reason, a detailed research is being done, as for the clustering part of Mahout, which uses the MapReduce paradigm, implemented by Hadoop."*

4.1 Apache Mahout: a data mining software

The Mahout project started by some people who were involved in the apache Lucene project who were interested in the fields of machine learning and had a desire for implementations with high scalability of some common machine learning algorithms for clustering and classification. Mahout began its life in 2008 as a subproject of apache's Lucene project, which provides advanced implementations of search, text mining and information-retrieval techniques. In the computer science field, these concepts had a lot in common with machine learning techniques like clustering or classification. As a result, by some people there was given a biggest interest in the machine learning areas and soon after, Mahout absorbed another open source collaborative filtering project, called as Taste. Some years later, Mahout became a top-level apache project in its own right.

Mahout, as we have mentioned, is a set of machine learning Java libraries which have been developed for various tasks, such as classification, clustering or pattern-mining. Much of Mahout's work has been not only implementing algorithms in an efficient and scalable way, but also has to do with the conversion of some of the most well-known algorithms to be able to work on top of Hadoop. Hadoop's logo is a yellow elephant, which explains the project name. A *mahout* is a person who keeps and drives an elephant.

The power of Mahout is hidden in the fact that the algorithms it implements are developed in such a way, in order to be run in a Hadoop environment. As we have already discussed, Hadoop is a distributed framework that allows for an algorithm to run concurrently on multiple nodes using the distributed computational and distributed storage techniques. The main idea behind Hadoop is having multiple machines that are able to handle the computational and storage task of very large datasets at the same time, referring to datasets whose records require an order in the scale of millions of records [67], [68], [69].

However, one could say that there are many user-friendly frameworks, equipped with more algorithms to perform these machine learning tasks. So why should we use Mahout instead of the other frameworks? Well, the main reason is that all the previous frameworks have not been developed to be *scalable* for very large datasets.

In short, Mahout aims to:

- Build and support a community of users, making it easy for everyone to do its tests and experiments with it
- Focus on real-world and practical use cases
- Provide quality documentation, with easy to understand example

Although Mahout is a project open to implementations of all kinds of machine learning techniques, currently it focuses on three specific areas of machine learning. These specific machine learning areas that Mahout currently supports its algorithms are mostly used in real applications. These areas are the following [69], [70]:

- Collaborative filtering (Recommender engines)
- Classification (Categorization)
- Clustering

Collaborative Filtering (Recommender engines)

Collaborative filtering (CF) is a technique that uses user information and behavior such as ratings, clicks, history and purchases to provide recommendations to other site users, with similar behavior. It is mostly used to recommend products to buy such as books, music and movies, but it is also used in other applications in order to shrink and sort multiple forms of data. Given a set of users and items, collaborative filtering applications recommend products to the current user of the system. There are four different ways of generating this kind of recommendations:

- *User-based*: Recommend items by finding users with similar behavior or preferences.
- *Item-based*: Calculate the similarity between similar products and provide users with different kind of recommendations.
- *Slope-One*: A simple and fast way of recommendation where different products are recommended according to user ratings
- *Model-based*: Provide recommendations according to a model which is developed according to users' behavior and ratings.

All the aforementioned approaches have the idea of calculating the similarity between users and their ratings. There are many ways to compute this kind of similarity while there are multiple measures so that you can determine which one works best for every kind of data.

Nowadays, there are hundreds of services or sites that try to recommend products such as books or movies, according to the web user's history and behavior. Some of these, are the following:

- *Facebook* uses recommender techniques to identify people which someone probably knows, but does not know if he has a Facebook account or not.
- *Amazon* is the most famous e-commerce which uses recommendations. Based on past purchases and the user's activity, it recommends items that is most likely to interest the user, but does not know it yet.
- *Netflix* recommends movies that may be interesting for the users, based to the ratings of the users with similar movie tastes and behavior.
- *Dating sites* can recommend people to people, according to the preferences of each one.

Classification

Classification techniques can find how much a thing belong or not to a category, or how much it does or does not have a specific attribute. Usually, these systems learn by looking at multiple f items which belong to various categories, in order to result in classification rules.

The goal of classification is to label unseen documents, so to sort them together. Many classification approaches in machine learning calculate a variety of statistics that combine the specifications of a document with a certain label, in order to create a model that can be used afterwards in order to classify unknown or unseen documents.

The mostly used features for classification usually can contain words or probabilities for these words to belong in a book. However, these features can be anything that helps to associate a document with a specific label and can be integrated into the classification algorithm.

In general, classification helps someone to decide whether a thing is similar to a previously observed pattern or not, and it is mostly used to classify behavior or patterns that seem not to be usual. For instance, classification could be used to detect fraudulent activities, or it could be used to understand the content of a message. Whether it indicates anger or satisfaction. One should have in mind that, each of these techniques work in a better way when they are provided with massive data, as the results are usually better. Sometimes, these techniques must not only work on massive amounts of data, but have to produce fast results, making the idea of scalability a very important issue.

Mahout offers various classification algorithms, most of which are developed to run on top of Hadoop. For these algorithms, in order to function properly, a model has to be

trained to represent the patterns to be identified, and then tested against a subset of the data set. In most cases of classification problems, one or more people have usually to go through and manually find a specific subset of the data, which will be used to train the classification algorithm.

Nowadays, this general idea has many applications among the various enterprises and companies. Some of these, are the following:

- *Yahoo! Mail* is able to decide whether or not an email is spam based on prior emails and spam reports from other users, as well as on characteristics of the email itself.
- *Google's Picasa*, a photo management application, is able to decide whether a region of an image contains a human face, according to the colors and the characteristics.
- *Optical character recognition* software is able to classify small regions of scanned text into specific and individual characters.
- *Apple's Genius* feature in iTunes, uses classification in order to classify songs into playlists that will be probably likable by its users.

Clustering

Clustering, as we have already seen, tries to group a large number of things together into multiple clusters that have similarities among them. It is used in order to discover hierarchy and order in a large or difficult-to-interpret data set. So it is used for finding interesting and previous unknown patterns and making the data set easier to understand.

Like CF, clustering calculates the similarity between items in a specific collection, but it only has to sort together similar items, in the same group of clusters. In many implementations of clustering, items are usually represented as vectors in a multi-dimensional space. As a result of this, one is able to calculate the distance between two items, if he has in his position the vectors of the data points and a specific distance measures. After that, the actual clusters can be calculated by grouping together the items that are close in distance.

As with classification, Mahout has various clustering algorithms, each with different characteristics. For instance, the kMeans algorithm is scalable but requires the user to specify the number of the initial clusters, while Dirichlet clustering requires the user to pick a distribution model, including the number of initial clusters he prefers.

There are many different ways to calculating the clusters, which have both advantages and disadvantages. Some of these ways work from the bottom up, by creating larger clusters from smaller ones, while other tend to partition a large cluster into smaller

ones. Both ways have various criteria for exiting the process at some point before they conclude to a pattern that is not interpretable and understandable.

Some examples where clustering is used, nowadays, to cluster unknown and unlabeled data are the following:

- *Google News* groups articles by topic using clustering techniques, in order to present news grouped by logical story, rather than presenting a full list of all the articles that have been written that day.
- *Search engines* group their search results using again their logical content, like before.
- In *companies*, consumers can group into different clusters according to several attributes such as their total income, geographical location and buying habits.

In Table 4, we can see the algorithms that Mahout supports, as for making recommendations, classification and clustering among very large datasets. For each of these algorithms, it is given a short description [67], [68], [69], [70].

Algorithm	Category	Description
Distributed Item-based Collaborative Filtering	Recommender Engines	Finds a user's preference for one item by looking at past preferences for similar items
Collaborative Filtering Using a Parallel Matrix Factorization	Recommender Engines	Predicts which items a user might prefer, among them which has not yet seen or bought
Naïve Bayes	Classification	Used to classify objects into binary categories
Hidden Markov Models	Classification	The system being modeled is assumed to be a Markov process with hidden states
Logistic Regression	Classification	Is used for prediction of the probability of an event to happen, using predictor variables that may be either numerical or categories
Random Forest	Classification	MapReduce implementation where each mapper builds a subset of the forest using only the data available in its partition
Canopy Clustering	Clustering	For preprocessing data before using a kMeans or Hierarchical clustering algorithm

Fuzzy kMeans	Clustering	Discovers soft clusters where a particular point can belong to more than one cluster, according to its degree of association to each cluster
kMeans Clustering	Clustering	Aims to partition **n** observations into **k** clusters in which each observation belongs to the cluster with the nearest mean.
Spectral Clustering	Clustering	Makes use of the spectrum of the similarity matrix of the data examining the connectedness of the data

Table 4 - Mahout's algorithms

In this thesis, as it was mentioned in the previous chapters, the algorithm which is going to be implemented to run in parallel on top of an OLAP database, is the fuzzy kMeans algorithm. This algorithm, as we can see in Table 4, has to do with the clustering techniques of Mahout, and for that reason more details about clustering with Mahout in general, are going to be given in *section 4.2*.

4.2 Clustering with Mahout

Mahout, as we have already discussed, supports several clustering-algorithm implementations, all written in the MapReduce framework, each with its own set of goals and criteria.

In general, the steps involved in clustering data using Mahout (Figure 19) are [69], [70]:

1. Prepare the input dataset. If clustering has to be performed with text, we have to convert the text to a numeric representation.
2. Run the clustering algorithm using one of the many Hadoop-ready driver programs that are available in Mahout.
3. Read and evaluate the results from the output directory.
4. Run again the experiment, if it is considered necessary.

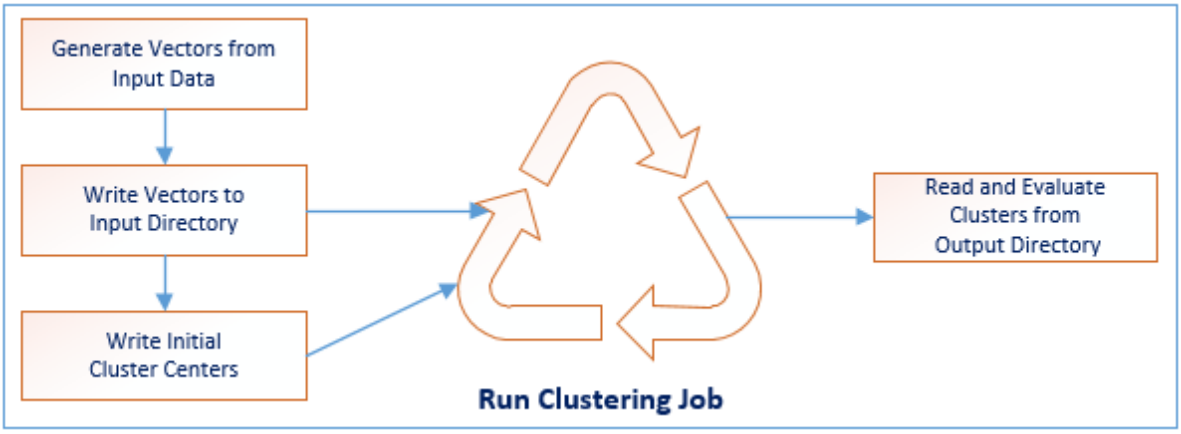

Figure 19 - Clustering with Mahout

Preprocess the data

Firstly, one should have in mind that clustering algorithms require data that is in a suitable format for processing. Generally, in Machine Learning, the data is often represented in a *vector* format, sometimes called a *feature vector*. In clustering, a vector is an array of weights that represent the data point and can come from various areas, such as a sensor data or user profiles.

In Mahout, vectors are implemented as three different classes, each of which is optimized for different scenarios: *DenseVector*, *RandomAccessSparseVector* and *SequentialAccessSparseVector*.

- *DenseVector* can be thought of as an array of doubles, whose size is the number of features in the data.
- *RandomAccessSparseVector* is like a HashMap between an integer and a double, where only non-zero valued features are permitted.
- *SequentialAccessSparseVector* is like having two parallel arrays, one of integers and the other of doubles, where again only non-zero valued entries are allowed.

Depending on the data, we will need to choose an appropriate implementation in order to have good algorithm performance. Generally speaking, text-based problems are sparse, making *SparseVector* the correct choice for them. On the other hand, if the values for most of the vectors are non-zero, then a *DenseVector* is more appropriate.

After that, we have to save the vectors in a *SequenceFile* format as input for the algorithm, which is a format from the Hadoop library that encodes a series of <key, value> pairs.

Run the Clustering algorithm

Finally the desired clustering algorithm is run, according to the different criteria and needs. Depending on what clustering algorithm is being chosen, the user has to set the *initial number of clusters*, the *maximum number of iterations* that the algorithm needs to run before stopping, a possible *threshold* after which the algorithm has to stop, a specific *distance measure*, according to which the distances between the data points and the cluster centroids are calculated, and so on.

As it was mentioned earlier, the clustering algorithms that Mahout implements run as MapReduce jobs, so according to how large the dataset is and how it has been split, the number of the mappers and the reducers is not predefined.

Read and evaluate the results

After all of these, the algorithm stops, it stores the results of each iteration to different sequence files and the user can observe and evaluate them, using the clusterdump utility of Mahout, which converts the sequence files to a human readable format. The *ClusterDumper* takes the input set of clusters and an optional dictionary that was generated during the conversion of the data to vectors, and outputs the clustering results into human readable form.

There are many approaches to *evaluating* the cluster results. Many people start simply by using manual inspection and ad-hoc testing. However, it is often necessary to use more in-depth evaluation techniques, as analyzing the output of clustering is an important steps to understand the output patterns. It can be done with simple

command-line tools or GUI-based visualizations. By the time that the clusters are visualized and the areas with the problems have been identified, these results can be transformed into quality measures, which give numeric values showing how good or how bad the final clusters are.

One good way for evaluating the clustering results is using the *inter-Cluster* and *intra-cluster distances*. In short, the distance between all pairs of cluster centroids can be calculated using some distance measure and can be represented in a table.

- This table of the *inter-Cluster distance* given the information of what happened after the clustering, by showing how near or far the resulting clusters ended up from each other.
- *Intra-Cluster distance* has to do with the distance among the data points that belong to a specific cluster, rather than the distance between two or more different cluster centroids. This kind of distance is able to give an idea of how well the distance measure was able to bring the items together.

If the results were not satisfying, then the process has to be repeated from the beginning, changing some of the parameters used to run the algorithm, or even changing the algorithm that was run.

Chapter 5

Parallel clustering with fuzzy kMeans

Summary: "*In this chapter the clustering algorithm which we are going to implement on top of an OLAP database, is explained in details. Moreover, the implementation of the aforementioned algorithm is being detailed, as for how it works with Mahout on top of Hadoop. Finally, the implementation that has been developed is mentioned in deep details, as for how the fuzzy kMeans clustering algorithm works in parallel on top of an OLAP database.*"

5.1 Fuzzy kMeans: A clustering algorithm

In fuzzy clustering, every point has a degree of belonging to a cluster, rather than belonging to just a single cluster. For that reason, points that are found to the edge of a cluster, may have a smaller degree of association to that cluster, than the points that can be found in the center of the cluster. *Fuzzy kMeans generally tries to deal with the problem where points are somewhat in between centers by replacing distance with probability, which could be some distance function, such as having probability relative to the inverse of the distance* [77].

Fuzzy kMeans belongs to the family of fuzzy logic based clustering algorithms and was introduced in 1984 by Bezdek. *It attempts to partition a collection of n elements into a collection of K clusters by associating each gene with all clusters via a real valued vector of indexes* [72]. As the name says, the fuzzy kMeans clustering algorithm implements a fuzzy form of the kMeans clustering, whereas it tries to generate overlapping clusters from a data set. kMeans tries to find the clusters where each point of the data set belongs to one single (hard) cluster, whereas fuzzy kMeans discovers the soft clusters. In a soft cluster, as it was mentioned earlier, any point can belong to more than one cluster with a certain degree of association towards each. This degree has to do with the distance from the point to the centroid of the cluster [71], [72], [73], [76].

In addition, fuzzy kMeans has a parameter, *m*, called the fuzziness factor, and has values greater than 1. In general, Fuzzy kMeans interacts with all of the data set but instead of assigning vectors to the nearest clusters, it calculates the degree of association of the point to each of the clusters. Suppose for a vector, V, that d1, d2, ..., dk are the distances to each of the k cluster centroids. The degree of association (u1) of vector (V) to the first cluster (C1) is calculated as [71], [75]:

$$u_i = \frac{1}{\left(\frac{d_1}{d_1}\right)^{\frac{2}{m-1}} + \left(\frac{d_1}{d_2}\right)^{\frac{2}{m-1}} + \ldots + \left(\frac{d_1}{d_k}\right)^{\frac{2}{m-1}}}$$

In the same way, we can calculate the degree of association to the other clusters by replacing d1 with d2, d3, and so on. It is clear from the expression that m should be greater than 1, or else we would have Non-accepted-Number (NaN) results.

- If we choose the value of 2 for m, we will see that all the degrees of association for any data point sum up to one.
- If we choose a value close to 1, like 1.000001, for m, the Fuzzy kMeans algorithm behaves more like the kMeans algorithm as m is getting closer to 1.

If m increases, the fuzziness of the algorithm increases, having more overlapping [74], [75], [76].

Algorithm's steps

In short, the fuzzy kMeans algorithm's basic steps are shown below:

1. The algorithm selects k points as the initial cluster centers ("means"), which represent the initial group centroids.

2. For each data point of the data set, the degree of association is calculated for each different cluster centroid, using the formula that was mentioned above.

$$u_i = \frac{1}{\left(\frac{d_1}{d_1}\right)^{\frac{2}{m-1}} + \left(\frac{d_1}{d_2}\right)^{\frac{2}{m-1}} + \ldots + \left(\frac{d_1}{d_k}\right)^{\frac{2}{m-1}}}$$

   m is usually considered as m=2 and the distances which are represented by the symbol d_i are calculated according to the chosen distance measure and are the distances between each point and each of the clusters centroids.

3. When all the degrees of association have been calculated, each cluster center (centroid) is recomputed as the sum of the data points multiplied each by its degree of association to the cluster, divided by the sum of the degree of association of the points in the specific cluster.

   For example, assuming that we have k points, then the cluster centroid of the cluster C1 would be:

$$C_1 = \frac{u1*coord1 + u2*coord2 + \ldots + uk*coordk}{u1 + u2 + \ldots + uk}$$

4. Steps 2 and 3 repeat until the centroids converge according to a given threshold, or until the number of maximum iterations has reached its maximum value.

5.2 Fuzzy kMeans on top of HDFS

For the implementation of fuzzy kMeans on top of HDFS, it was used the projects of Mahout, which contains machine learning libraries which support the field of clustering. The design is similar to the other clustering algorithms in Mahout, as it was mentioned in *Chapter 4*.

The implementation of the algorithm, accepts as an input a sequence file which contains vector points. It should be mentioned that the user can either provide the cluster centers as input or can allow canopy algorithm to run and create the initial number of clusters.

For every iteration, the cluster output is stored in a directory named as clusters-N (N is the iteration number), while the program does not modify the input directories.

The code has set a number of reduce tasks equal to the number of map tasks. So, in every reduce task there are created files with the name of part-00000, which contain the results of the MapReduce jobs and are stored into the clusters-N directory (N is the iteration number).

The final results of the MapReduce jobs, are stored in a folder named as clusters-N-final (N is the iteration number), and the algorithm finishes its job when the maximum number of iterations has been reached or there is convergence, according to a specific threshold.

The code of the clustering Mahout implementation, in order to have results using the MapReduce paradigm, is using the classes of a *Driver*, a *Mapper*, a *Combiner* and a *Reducer* as follows [74]:

- *FuzzyKMeansDriver*: The *Driver* iterates over the input data points and the cluster centroids for the specified number of iterations or until it is converged. During every iteration, a new clusters-N directory is created which contains the modified cluster centers obtained during the Fuzzy kMeans iteration. This will be provided as input clusters in the next iteration. Once Fuzzy KMeans is run for specified number of iterations or until it is converged, a map task is run to output the point and the cluster membership to each cluster pair, as final output to a directory named as clusteredPoints.

  The class of the FuzzyKMeansDriver, in order to work properly, is fed with the following variables:

  1. *input*: a file path string to a directory that contains the input data, stored in a Sequence File format.

2. *clustersIn*: a file path string to a directory that contains the initial clusters, stored in a Sequence File format.
3. *output*: a file path string to an empty directory which is used for the output from the algorithm, to be stored.
4. *convergenceDelta*: a double value used to determine if the algorithm has converged or not.
5. *maxIterations*: the maximum number of iterations that the algorithm is able to run.
6. *m*: a double which is the "fuzzyness" factor.
7. *runClustering*: a boolean that indicates that if it is true, the clustering step will be executed after the clusters have been determined.
8. *emitMostLikely*: a boolean that indicates that if it is true, the clustering step should only emit the most likely cluster for each clustered point.
9. *Threshold*: a double indicating that if the emitMostLikely is false, then it is the cluster probability threshold which is used for emitting multiple clusters for each point.
10. *runSequential*: a boolean that indicates that if it is true, the algorithm will run sequentially in memory, rather that running MapReduces jobs.

- *FuzzyKMeansMapper*: The *Mapper* reads the input cluster and computes the degree of association of each point to each cluster. The degree of association has to do with the distance of the data point to the cluster centroid, which is computed using the user supplied distance measure.

- *FuzzyKMeansCombiner*: The *Combiner* receives all the <key, value> pairs from the Mapper and produces partial sums of the degrees of association of the data points, for each cluster.

- *FuzzyKMeansReducer*: The *Reducers* receive the keys and the values that are associated with those keys, and afterwards they sum the values to produce a new centroid for each cluster. Finally, the reducer encodes the clusters which have not converged with a 'SC' cluster ID and the clusters which have converged with a 'SV' cluster ID.

In the following figures, it is a given a full image of how the Fuzzy kMeans implementation works on top of HDFS. For each of the figures, it will be given an explanation of how the algorithm works and outputs its final results to the user, in order to observe and evaluate them.

Preprocessing steps

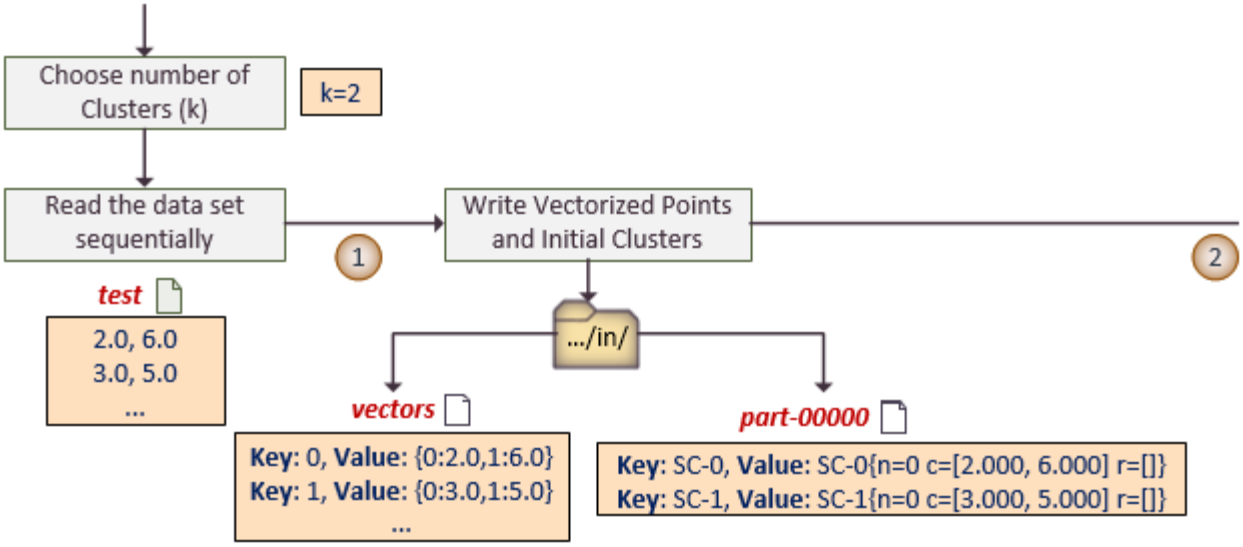

Figure 20 - Preprocessing steps on top of HDFS

Firstly, when the project is executed, it asks from the user to insert the number of desired clusters, in order the implementation to know how many clusters it is going to create.
Example: k=2.

After that, it reads the whole data set sequentially (line by line), and it stores it to the HDFS, splitting it in blocks of 64 MB each.
Example: We have a data set called **test,** which is stored on the HDFS in one block.

The implementation's first job is to create in the HDFS, the input folder, where it writes a new sequence file, which contains the data set in a vectorised form (<key, value> pairs), with different key for each value.
Example: It stores the sequence file called **vectors** which contains the data set in the following way:

Key: 0, **Value**: {0:2.0, 1:6.0}
Key: 1, **Value**: {0:3.0, 1:5.0}
…

In addition, it writes a sequence file which contains the initial centroids, which are the first *k* different points of the data set and have the *SoftKluster* format.
Example: It writes the sequence file called part-00000, which contains two (2) initial centroids in <key, value> pairs, with a different key for each value (SoftKluster):

Key: SC-0, **Value**: SC-0{n=0 c=[2.000, 6.000] r=[]}
Key: SC-1, **Value**: SC-1{n=0 c=[3.000, 5.000] r=[]}

Pre-MapReduce steps

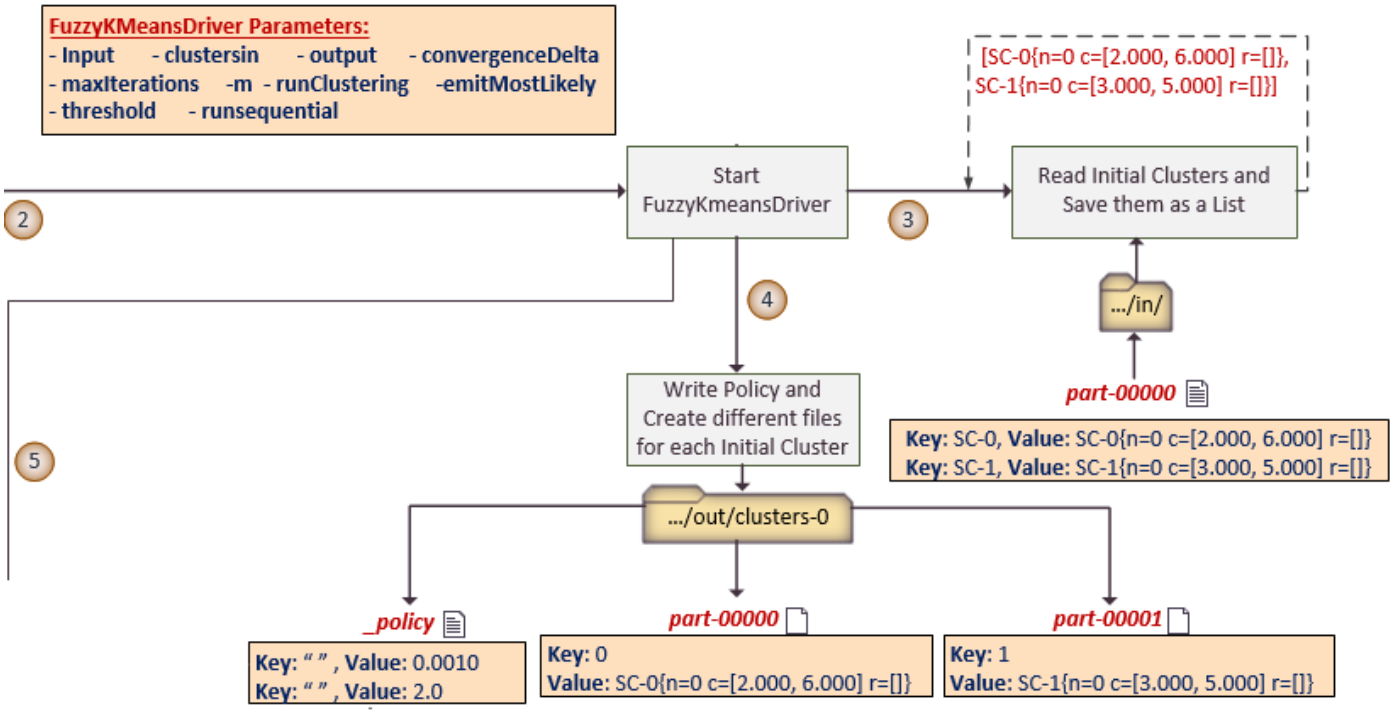

Figure 21 - pre-MapReduce steps on top of HDFS

The next job of the implementation has to do with starting the FuzzyKmeansDriver, which takes as an input the parameters we have explained above: *Input, clustersin, output, convergenceDelta, maxIterations, m, runClustering, emitMostLikely, threshold, runsequential.*

Example: The FuzzyKmeansDriver accepts the following parameters:

Input: vectors	m: 2
Clustersin: part-00000	runClustering: true
Output: out	emitMostLikely: true
convergenceDelta: 0.005	threshold: 0.0
maxIterations: 20	runsequential: false

But firstly, it reads the sequence file which contains the initial centroids in <key, value> pairs, and it returns the values as a list.

Example: Afterwards, it reads the part-0000 sequence file and it returns the following list:

[SC-0{n=0 c=[2.000, 6.000]=[]}, SC-1{n=0 c=[3.000, 5.000] r=[]}]

Then, it creates the folder named as clusters-0, where it stores a sequence file called *_policy* which contains the value of the *convergence delta* and the value of the number **m**. In addition, it stores *k* files which contain the *k* centroids, in <key, value> pair format. After all of these, it is time for the MapReduce jobs to start.
Example: The FuzzyKMeansDriver runs and stores to the HDFS, the **_policy** file which contains the following:

Key: " " , Value: 0.0010
Key: " " , Value: 2.0

Finally, it creates and stores to the out folder of the HDFS, two (2) different sequence files named as part-00000 and part-00001, which are the following:

Key: 0, Value: SC-0{n=0 c=[2.000, 6.000] r=[]}
Key: 1, Value: SC-1{n=0 c=[3.000, 5.000] r=[]}

MapReduce steps

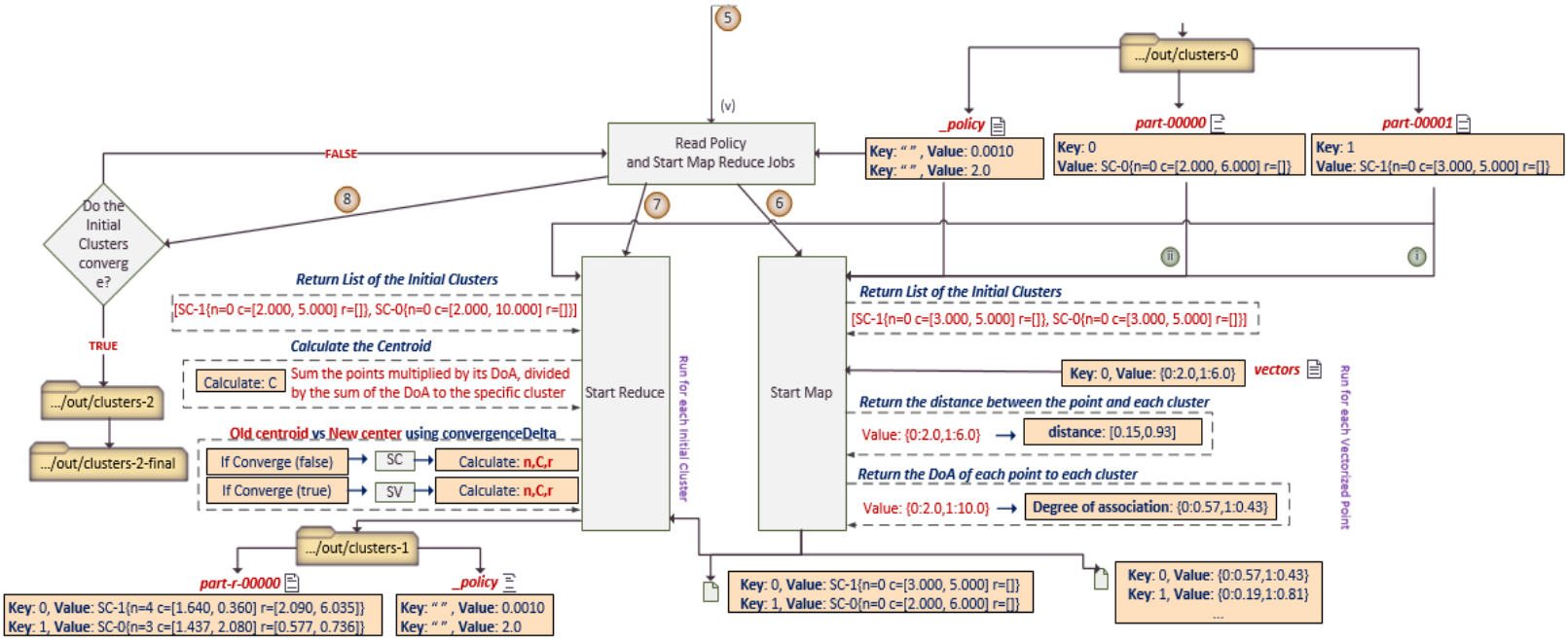

Figure 22 - MapReduce on top of HDFS

Before the Map job starts, the Fuzzy kMeans implementation reads the contents of the *_policy* file which contains the convergence delta and the m.
Example: It reads the _policy file which contains the following:

Key: " " , Value: 0.0010
Key: " " , Value: 2.0

The Map phase starts. Firstly, it reads the *k* sequence files which contain the initial centroids in <key, value> pairs and returns a list with the values of the *k* centroids.
Example: It reads the part-0000 and part-00001 sequence files and it returns the following list, with the values of the <key, value> pairs, with the initial centroids:

[SC-0{n=0 c=[2.000, 6.000]=[]}, SC-1{n=0 c=[3.000, 5.000] r=[]}]

Then, *for each vectorised point* the map job, it is doing the following. It returns the distance between each vectorised point and each cluster centroid and it stores it in a list.
Example: During the first time that the Map job runs, it uses the Euclidean distance measure and it calculates the Euclidean distance between the point with value {0:1.0,1:1.0} and the cluster centroids [2.000, 6.000] and [3.000, 5.000]. The list that it returns is the following:

[5.099, 4.472]

The same happens for all the other points.

Afterwards, having calculated the aforementioned distances, it uses the following formula to calculate the Degree of Association of each point to each cluster, and it stores it in vector format.

$$u_i = \frac{1}{\left(\frac{d_1}{d_1}\right)^{\frac{2}{m-1}} + \left(\frac{d_1}{d_2}\right)^{\frac{2}{m-1}} + \ldots + \left(\frac{d_1}{d_k}\right)^{\frac{2}{m-1}}}$$

Finally, it stores the results of the degree of association of the different points of the data set, in <key, value> pairs, after having scanned the whole sequence file with the data points.
Example: During the first time that the Map job runs, it calculates the degree of association of the point with value {0:1.0, 1:1.0} and the clusters with the following centroids [2.000, 6.000] and [3.000, 5.000]. The vector that it returns is the following:

{0:0.4347, 1:0.5652}

The same happens for all the other points.

After that, it stores the results of all of the degrees of association of all the points:

Key: 0, Value: {0:0.57,1:0.43}
Key: 1, Value: {0:0.19,1:0.81}
Key: 2, Value: {0:0.4347,1:0.5652}
...

Then, it is time for the Reduce jobs to start.

Firstly, it reads the *k* sequence files which contain the initial centroids in <key, value> pairs and returns a list with the values of the *k* centroids.
Example: It reads the part-0000 and part-00001 sequence files and it returns the following list, with the values of the <key, value> pairs, with the initial centroids:

[SC-0{n=0 c=[2.000, 6.000]=[]}, SC-1{n=0 c=[3.000, 5.000] r=[]}]

Then, *for each initial cluster*, it calculates the new cluster centroid by adding all the data points together, multiplied by their degree of association to each cluster, and dividing this sum with the sum of the degrees of association.
Example: It calculates that the new cluster centroid of the 1st cluster with key equal to **0** is:

c=[1.640, 0.360]

It compares the new calculated centroid with the old centroid using the convergence delta that was assigned in the beginning. If their difference is less or equal than the convergence delta, then the centroids converge, so the final form of the cluster changes from SC to SV. On the other hand, the final form of the cluster remains as is.
Example: It calculates the Sqrt(1.64^2+0.36^2)-2*(1.64*2+0.36*6)+(2^2+6^2)) and compares it with the convergence delta for the cluster with key equal to **0**, the result shows that they do not converge, so the final form of the cluster that will be calculated is SC.

Finally, the reduce phase calculates the following as for each cluster:

- The total number of the points that belong to the cluster, by adding all the degrees of association of the points to the cluster
- The new centroid, by adding all the data points together, multiplied by their degree of association to each cluster, and dividing this sum with the sum of the degrees of association.
- The new radius (population standard deviation), by calculating the square root of the data points multiplied twice by their degree of association to each cluster, multiplied with the total number of points and extracted by the square of the sum of the new centroid. This square root, divided by the total number of the points that belong to the cluster, is the radius of the aforementioned cluster.

The results are written in a new clusters folder, into a sequence file (<key, value> pairs), as well as the _policy file which is stored again to the new clusters folder.

Example: The implementation after doing the aforementioned calculations, it calculates the following for the cluster with key equal to **0**.

- n = n1+n2+… = 4
- c_x = (n1*u1+n2*u2+…)/n = 1.640
- rx = Sqrt.[(n1*u1*u1+n2*u2*u2+…)*n - cx^2]/n = 2.090

The final cluster has the following format:

SC-1{n=4 c=[1.640, 0.360] r=[2.090, 6.035]}

It is stored in the folder clusters-1 in the HDFS, in the sequence file part-r-00000 which contains <key, value> pairs for each cluster.

Key: 0, Value: SC-1{n=4 c=[1.640, 0.360] r=[2.090, 6.035]}
Key: 1, Value: SC-0{n=3 c=[1.437, 2.080] r=[0.577, 0.736]}

Moreover, in the folder clusters-1, it is stored the _policy file which contains the following:

Key: " " , Value: 0.0010
Key: " " , Value: 2.0

However, during the check of the convergence, if all the centroids converge with their old-ones at some iteration, then the clusters folder where the converged clusters are stored, is renamed to clusters-final folder, and stores the final results.
Example: The cluster converged at the second iteration, so the folder *clusters-2* with the converged clusters, is renamed to *clusters-2-final*.

runClustering set to true

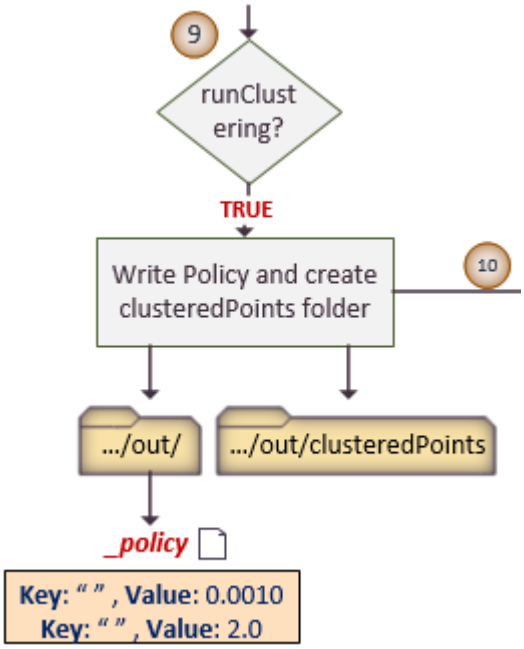

Figure 23 - runClustering set to true on top of HDFS (1/2)

In the case that the boolean variable of runClustering at the FuzzyKmeansDriver is set to true, then it is time for one last map job to run, in order for the algorithm to show where and how the data points have been assigned to each cluster. In that case, a folder named as clusteredPoints is created, while the ***_policy*** sequence file is stored to the output folder of the *HDFS*.
Example: The folder clusteredPoints is created and the _policy file is stored to the folder **out** of the HDFS, including the following:

Key: " " , Value: 0.0010
Key: " " , Value: 2.0

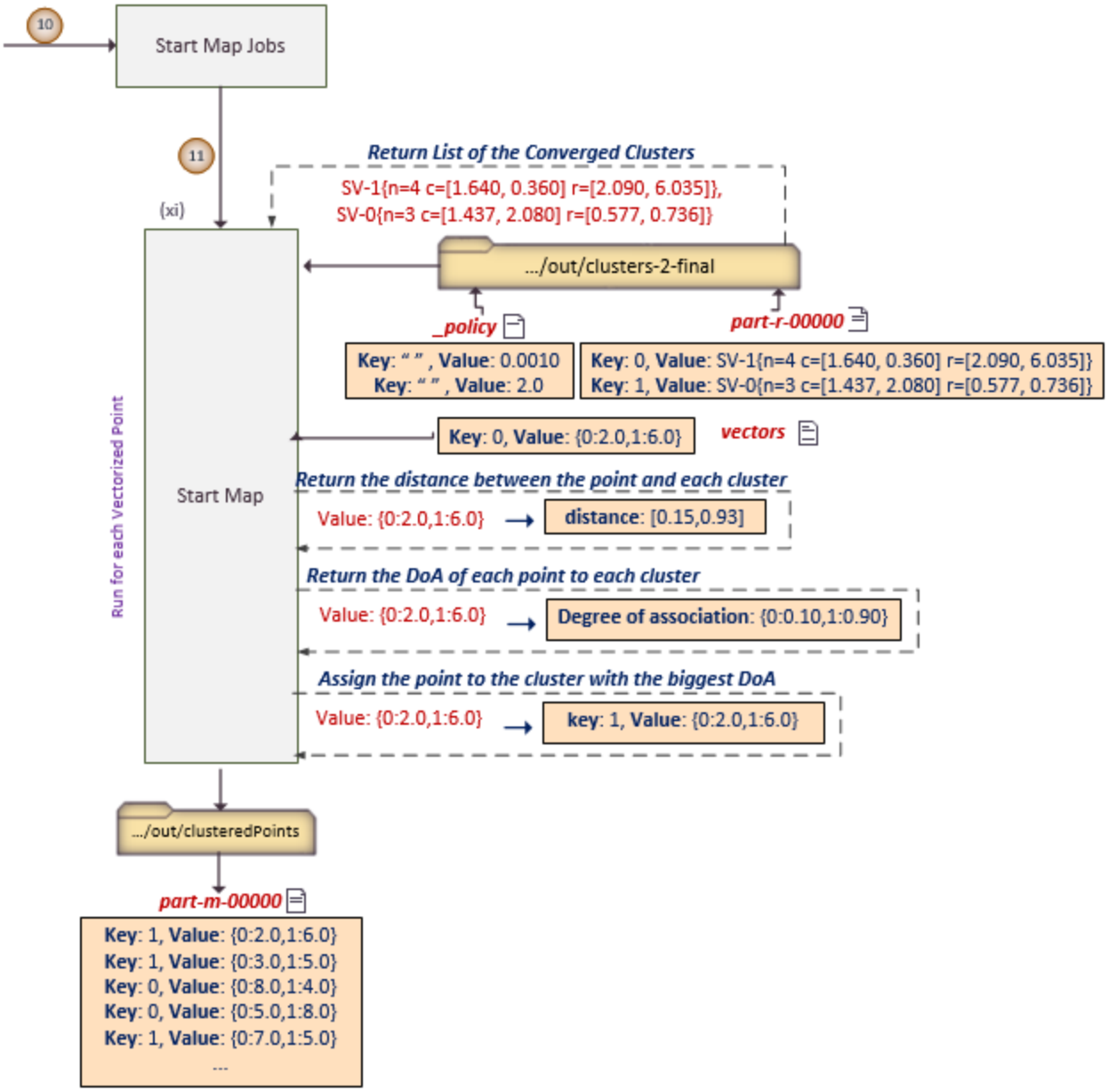

Figure 24- runClustering set to true on top of HDFS (2/2)

Then, the final Map phase takes place.

It reads from the clusters final folder the _policy file and the sequence file of the converged clusters, and returns their values as a list.
Example: It reads the part-00000 sequence file of the clusters-2-final folder and it returns the following list, with the values of the <key, value> pairs, with the initial centroids:

SV-1{n=4 c=[1.640, 0.360] r=[2.090, 6.035]}, SV-0{n=3 c=[1.437, 2.080] r=[0.577, 0.736]}

After that, for each Vectorised point it calculates the distance between the point and each of the cluster centroids and it stores it in a list.
Example: During the first time that the Map job runs, it uses the Euclidean distance measure and it calculates the Euclidean distance between the point with value {0:1.0,1:1.0} and the cluster centroids [1.640, 0.360] and [1.437, 2.080]. The list that it returns is the following:

[0.905, 1.165]

The same happens for all the other points.

Afterwards, having calculated the aforementioned distances, it uses the following formula to calculate the Degree of Association of each point to each cluster, and it stores it in vector format.

$$u_i = \frac{1}{\left(\frac{d_1}{d_1}\right)^{\frac{2}{m-1}} + \left(\frac{d_1}{d_2}\right)^{\frac{2}{m-1}} + \ldots + \left(\frac{d_1}{d_k}\right)^{\frac{2}{m-1}}}$$

Finally, it assigns the data point to the cluster with the biggest degree of association, storing it in a <key, value> pair format, where the key is the cluster identifier and the value is the vectorised format of the data point.
Example: During the first time that the Map job runs, it calculates the degree of association of the point with value {0:1.0, 1:1.0} and the clusters with the following centroids [1.640, 0.360] and [1.437, 2.080]. The vector that it returns is the following:

{0:0.623, 1:0.3763}

As we can see, the point has a bigger degree of association with the cluster with identifier equal to 0, so it is assigned that it belong to this cluster, by creating the following <key, value> pair:

key: 0, Value: {0:1.0,1:1.0}

Finally, these <key, value> pairs are stored in a sequence file format to the clusteredPoints folder of the HDFS, making the algorithm to stop running.
Example: The sequence file is named as *part-m-00000* and is stored to the folder clusteredPoints. By reading this sequence file, we could see how the data points have been assigned to each cluster, according to their keys.

Key: 1, Value: {0:2.0,1:6.0}
Key: 1, Value: {0:3.0,1:5.0}
Key: 0, Value: {0:1.0,1:1.0}
Key: 0, Value: {0:8.0,1:4.0}
...

5.3 Fuzzy kMeans on top of an OLAP database

The implementation of fuzzy kMeans on top of an OLAP database, has many differences with the implementation that was explained in the *section 5.2*.

In general, there are two main differences:

- The idea of HDFS does not exist anymore. All the aforementioned files that were created and stored in different folders and sub-folders of the HDFS, are now stored entirely in an OLAP database.

- The MapReduce paradigm has been fully replaced by a different <key, value> pair based implementation. In short, multiple threads read and write in parallel (concurrently) different data on the database tables, and do the same job without interrupting each other. At each thread, it is assigned a specific area of data where it can work, without having to communicate with the other threads or wait for the other threads to finish their jobs.

More details about how the fuzzy kMeans implementation works in parallel on top of an OLAP database, will be given and explained accompanied with the following figures. It must be mentioned that in these figures, the differences between the previous implementation are shown clearly, marked with a red "X" symbol.

Preprocessing steps

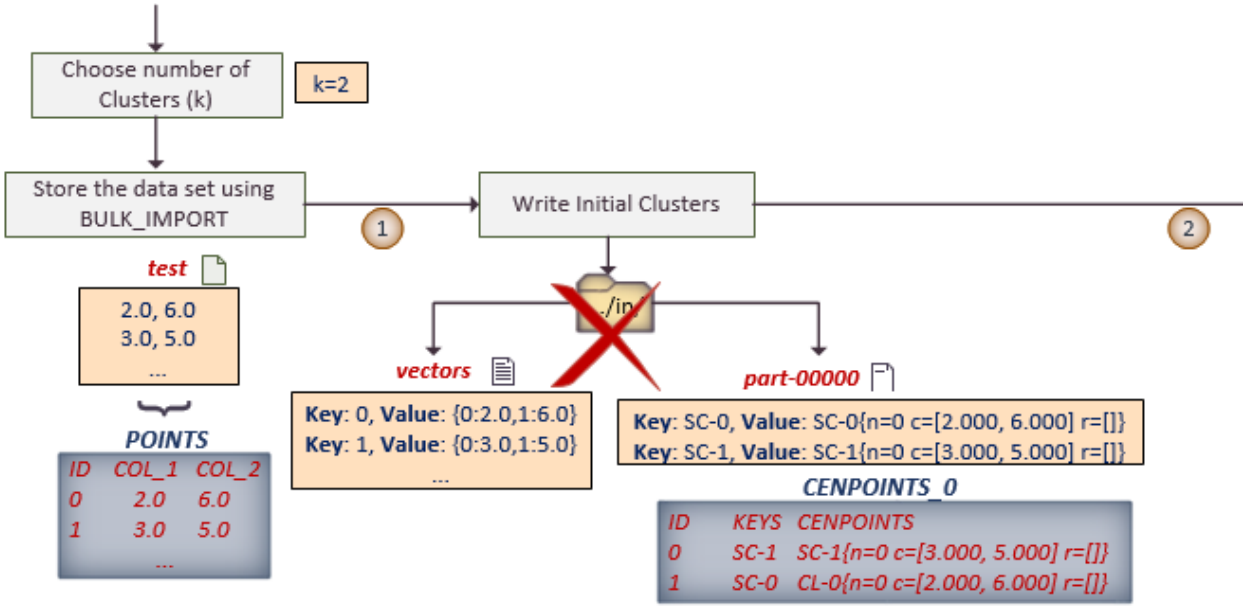

Figure 25 - Preprocessing steps on top of an OLAP database

Firstly, when the project is executed, it asks from the user to insert the number of Clusters, in order for the Fuzzy kMeans implementation to know how many clusters it is going to create.
Example: k=2.

After that, it stores the whole data set, using the BULK_IMPORT option where the implementation creates a database table with as many columns as the coordinates of the data points of the data set, and after that it writes the whole file, by splitting it between the commas, and writing these different comma separated parts to the table.
Example: We have a data set called *test*, which is stored in the table *POINTS*, like we can see below:

ID	COL_1	COL_2
0	2.0	6.0
1	3.0	5.0
	...	

The implementation's first job is to store the initial clusters to a database table, which has as many rows as the number *k*.
Example: Because the initial clusters are two (2), it inserts as the clusters' centroid the first two data points of the data set, in the table *CENPOINTS_0*:

ID	KEYS	CENPOINTS
0	SC-1	SC-1{n=0 c=[3.000, 5.000] r=[]}
1	SC-0	CL-0{n=0 c=[2.000, 6.000] r=[]}

Pre-NewMapReduce steps

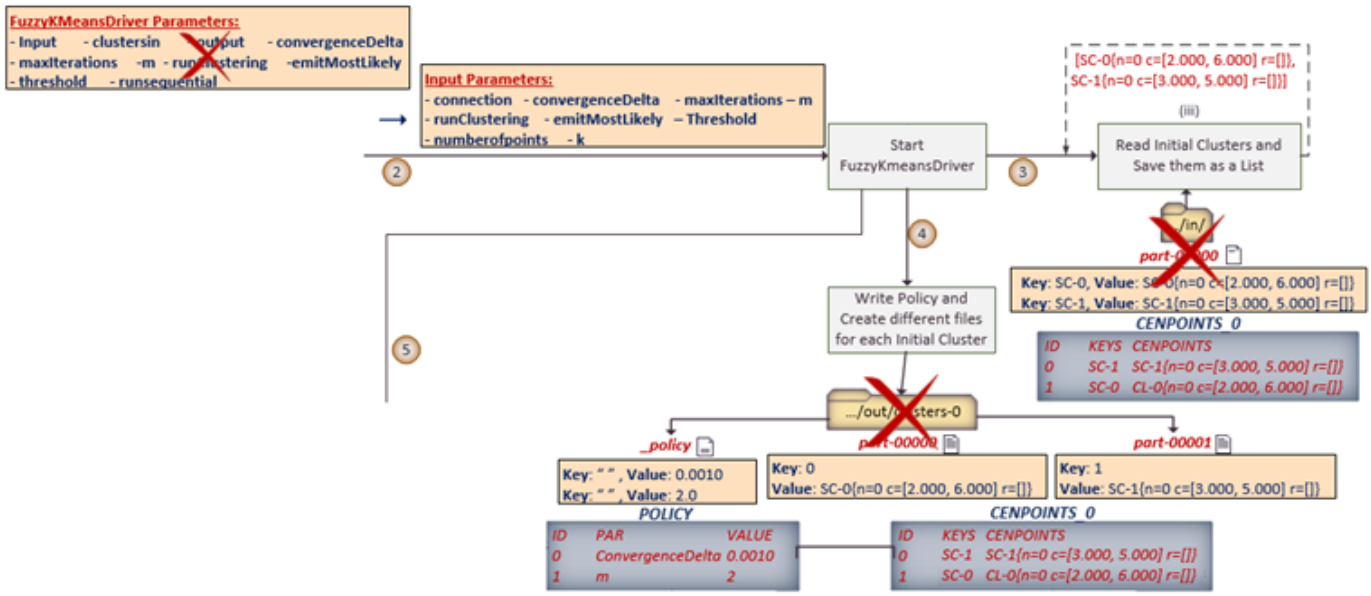

Figure 26 - pre-NewMapReduce steps on top of an OLAP database

The next job of the implementation has to do with starting the FuzzyKmeansDriver, which takes as an input the following parameters: *connection, convergenceDelta, maxIterations, m, runClustering, emitMostLikely, threshold, numberofpoints, k.*
Example: The FuzzyKmeansDriver accepts the following parameters:

connection: database connect	emitMostLikely: true
convergenceDelta: 0.005	threshold: 0.0
maxIterations: 20	Numberofpoints: 8
m: 2	k: 2
runClustering: true	

But firstly, it reads the database table which contains the initial centroids in <key, value> pairs, and it returns the values as a list.
Example: Afterwards, it reads the table *CENPOINTS_0*

ID	KEYS	CENPOINTS
0	SC-1	SC-1{n=0 c=[3.000, 5.000] r=[]}
1	SC-0	CL-0{n=0 c=[2.000, 6.000] r=[]}

and it returns the following list:

[SC-0{n=0 c=[2.000, 6.000]=[]}, SC-1{n=0 c=[3.000, 5.000] r=[]}]

Then, it creates and writes to a database table the value of the *convergence delta*, and the value of the number *m*. After all of these, it is time for the New MapReduce jobs to start.
Example: The FuzzyKMeansDriver runs and creates the table *POLICY* with the values of the convergence delta and the m, as follows:

ID	PAR	VALUE
0	ConvergenceDelta	0.0010
1	m	2

NewMapReduce steps

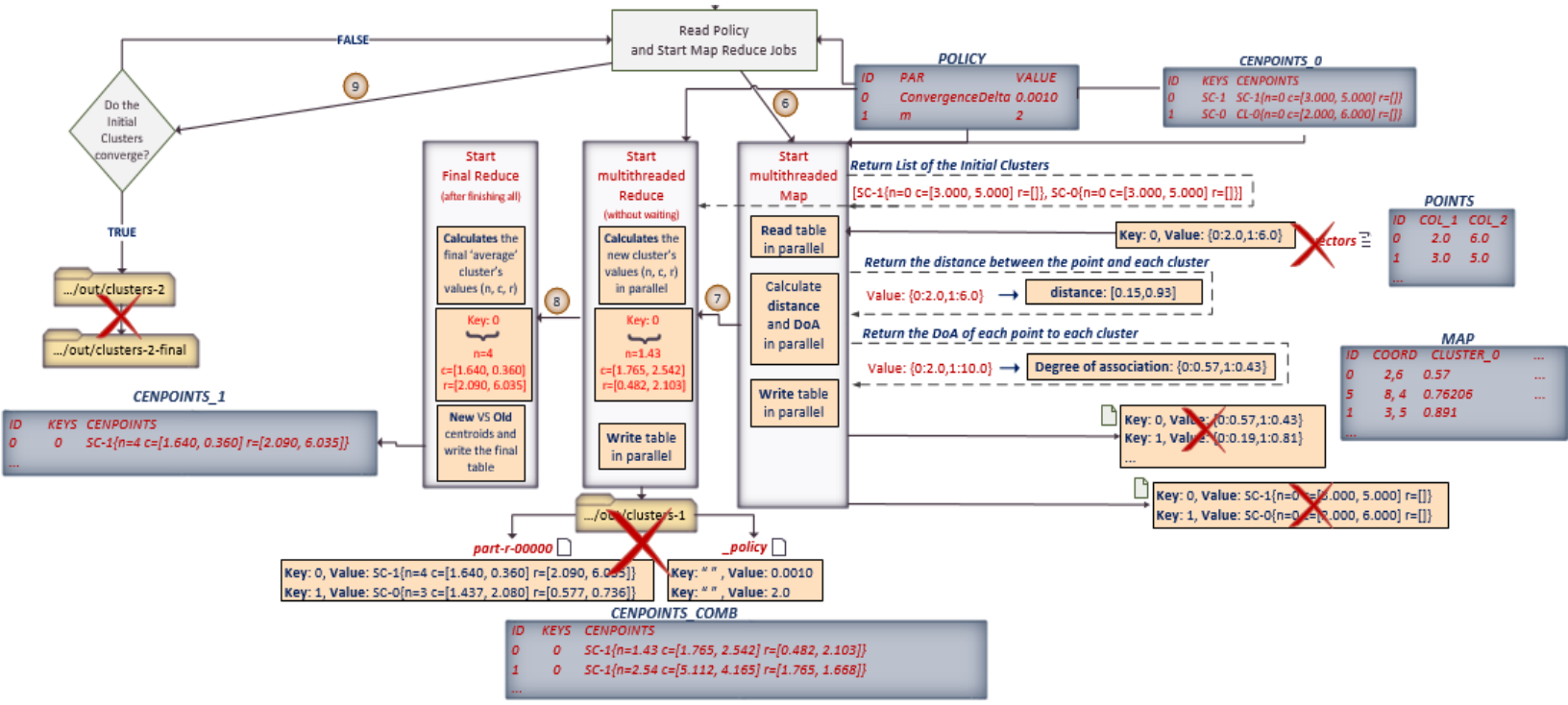

Figure 27 - NewMapReduce steps on top of an OLAP database

Before the multithreaded map job starts, the fuzzy kMeans implementation reads the contents of the database table with the policy values which contain the convergence delta and the m.
Example: It reads the table *POLICY* with the values of the convergence delta and the m, as follows:

ID	PAR	VALUE
0	ConvergenceDelta	0.0010
1	m	2

The *multithreaded Map phase* starts.

Before continuing explaining the whole process, it should be mentioned that in order for the new MapReduce jobs to run in parallel without using the MapReduce paradigm, we have used the concept of threads. More particularly, according to the number of the data points that are stored in the database table *POINTS*, there are created different amount of threads, for splitting and assigning to them, multiple New MapReduce jobs, in order to run in parallel. For instance, we have considered that each thread will handle 1000 different points and will run its own New MapReduce job, without being interrupted or waiting for the other threads to finish.

So if we had a data set with 5000 data points, there would be created 5 different threads, where each one would implement its own New MapReduce Job, with different data points each.

Threads	IDs of data points
Thread-1	1-1000
Thread-2	1001-2000
Thread-3	2001-3000
Thread-4	3001-4000
Thread-5	4001-5000

Table 5 - MapReduce jobs assigned to threads

As a result of all of these, various threads are created.

Firstly, each thread reads the database table which contains the initial centroids in <key, value> pairs and returns a list with the values of the *k* centroids. In addition, it reads the contents of the database table with the policy information.
Example: It reads the table *CENPOINTS_0:*

ID	KEYS	CENPOINTS
0	SC-1	SC-1{n=0 c=[3.000, 5.000] r=[]}
1	SC-0	CL-0{n=0 c=[2.000, 6.000] r=[]}

and it returns the following list, with the values of the <key, value> pairs, with the initial centroids:

[SC-0{n=0 c=[2.000, 6.000]=[]}, SC-1{n=0 c=[3.000, 5.000] r=[]}]

In addition, it reads the table *POLICY* which include the convergence delta and the number m:

ID	PAR	VALUE
0	ConvergenceDelta	0.0010
1	m	2

Then, *each thread* is doing the following. It reads the assigned to it values of the database table with the data points, and it returns the distance between each vectorised point and each cluster centroid, storing it into a list.
Example: The first thread (thread-1), reads the assigned to it values of the table POINTS:

ID	COL_1	COL_2
0	2.0	6.0
1	3.0	5.0
...		

Then, it uses the Euclidean distance measure and it calculates the Euclidean distance between the point with value {0:1.0, 1:1.0} and the cluster centroids [2.000, 6.000] and [3.000, 5.000].

The list that it returns is the following:

[5.099, 4.472]

The same happens for all the other points, for each different thread.

Afterwards, having calculated the aforementioned distances, each thread uses the following formula to calculate the degree of association of each of the assigned to it points, to each cluster, and it stores it in vector format.

$$u_i = \frac{1}{\left(\frac{d_1}{d_1}\right)^{\frac{2}{m-1}} + \left(\frac{d_1}{d_2}\right)^{\frac{2}{m-1}} + \ldots + \left(\frac{d_1}{d_k}\right)^{\frac{2}{m-1}}}$$

Finally, each thread stores the results of the degree of association of the different points of the data set in a database table, in a random order.
Example: The first thread (thread-1), calculates the degree of association of the point with value {0:1.0, 1:1.0} and the clusters with the following centroids [2.000, 6.000] and [3.000, 5.000].
The vector that it returns is the following:

{0:0.4347, 1:0.5652}

The same happens for all the other points, for each different thread.

After that, each thread stores the results of all of the degrees of association of all the points, accompanied with their identification number and their coordinates, in the table *MAP*:

ID	COORD	CLUSTER_0	...
0	2,6	0.57	...
5	8, 4	0.76206	...
1	3, 5	0.891	
...			

After one of the threads finish its map job, it is time for the Reduce jobs to start, without waiting the other threads to finish their own jobs.

To begin with, each thread reads the database table which contain the initial centroids in <key, value> pairs and returns a list with the values of the *k* centroids. In addition, it reads the contents of the database table with the policy information.
Example: It reads the table *CENPOINTS_0*:

ID	KEYS	CENPOINTS
0	SC-1	SC-1{n=0 c=[3.000, 5.000] r=[]}
1	SC-0	CL-0{n=0 c=[2.000, 6.000] r=[]}

and it returns the following list, with the values of the <key, value> pairs, with the initial centroids:

[SC-0{n=0 c=[2.000, 6.000]=[]}, SC-1{n=0 c=[3.000, 5.000] r=[]}]

In addition, it reads the table *POLICY* which include the *convergence delta* and the number *m*:

ID	PAR	VALUE
0	ConvergenceDelta	0.0010
1	m	2

Then, each thread, *for each initial cluster*, for its *assigned data points* calculates the following:

- The total number of the points that belong to the cluster, by adding all the degrees of association of the points to the cluster
- The new centroid, by adding all the data points together, multiplied by their degree of association to each cluster, and dividing this sum with the sum of the degrees of association.
- The new radius (population standard deviation), by calculating the square root of the data points multiplied twice by their degree of association to each cluster, multiplied with the total number of points and extracted by the square of the sum of the new centroid. This square root, divided by the total number of the points that belong to the cluster, is the radius of the aforementioned cluster.

The results are written in a database table which contains the new "middle" clusters, which will be averaged in the next step, of the final reduce, according to their common keys.

More particularly, each thread deals with the data points that have been assigned to it, like there are no other data points. For that reason, each thread create its own clusters, which we will have to merge, by combining the clusters with the same identification key.
Example: Each thread, calculates the following for the cluster with key equal to **0**.

- n = n1+n2+… = 1.43
- c_x = (n1*u1+n2*u2+…)/n = 1.765
- rx = sqrt.[(n1*u1*u1+n2*u2*u2+…)*n - cx^2]/n = 0.482

The "middle" cluster has the following format:

SC-1{n=1.43 c=[1.765, 2.542] r=[0.482, 2.103]}

Finally, the results are stored in the table ***CENPOINTS_COMB***, where each cluster is accompanied with its own key, in order to find afterwards the clusters with the common keys and merge them.

ID	*KEYS*	*CENPOINTS*
0	*0*	*SC-1{n=1.43 c=[1.765, 2.542] r=[0.482, 2.103]}*
1	*0*	*SC-1{n=2.54 c=[5.112, 4.165] r=[1.765, 1.668]}*
…		

However, in that part, in order the procedure to continue, we have to wait for all the threads to complete their New MapReduce jobs, as our final step has to do with the combination of clusters with the common keys, according to the database table *CENPOINTS_COMB*.

So, the time for the Final Reduce has come.
During that phase, for the clusters that we have in the middle table *CENPOINTS_COMB* with common keys, we calculate the total number of observations (n), the total value of the centers (c) and the total value of the radius (r).

- In order to calculate the total n, we add the number n of each centroid.
- In order to calculate the final c, we calculate for each centroid a number called S1, by multiplying the number n of each cluster, with its center and by adding the different values of S1. So, after all of these $c_x=S1_x/n$
- In order to calculate the final r, we calculate for each centroid a number called S2 by adding the squared radius multiplied with the squared n, with the squared S1 and by dividing the result with the number n. Finally, we add the different values of S2. So, after all of these $r_x = (sqrt(S2_x*n-S1_x*S1_x))/n$

Example: In our case, after the aforementioned computations, the merged cluster of the following clusters:

SC-1{n=1.43 c=[1.765, 2.542] r=[0.482, 2.103]}
SC-1{n=2.54 c=[5.112, 4.165] r=[1.765, 1.668]}

is: SC-1{n=4 c=[1.640, 0.360] r=[2.090, 6.035]}

The final merged clusters are stored in a database table, similar to the database table that had the initial clusters.
Example: The results are stored in the table called *CENPOINTS_1*, where number 1 indicates the iteration number.

ID	KEYS	CENPOINTS
0	0	SC-1{n=4 c=[1.640, 0.360] r=[2.090, 6.035]}
...		

Afterwards, the implementation selects the new centroids of the changed clusters from the database table and it compares them with the old ones, using the convergence delta which is selected from the database table with the policy variables. If their difference is less or equal than the convergence delta, then the centroids converge, so the final cluster ID changes from SC to SV. On the other hand, the form of the cluster remains as is.
Example: It calculates the Sqrt(1.64^2+0.36^2)-2*(1.64*2+0.36*6)+(2^2+6^2)) and compares it with the convergence delta for the cluster with key equal to **0**, the result shows that they do not converge, so the final form of the cluster that will be calculated is SC.

runClustering set to true

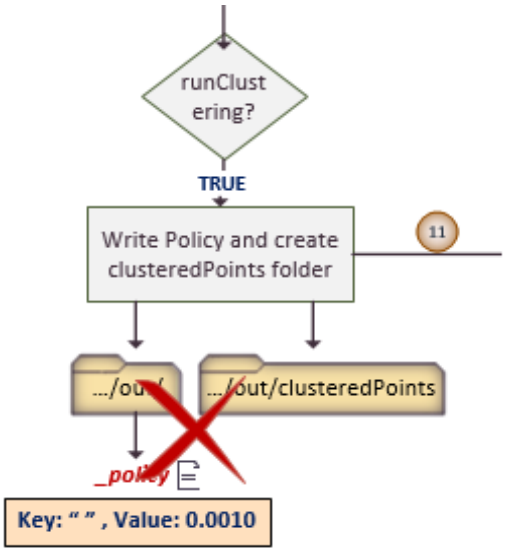

Figure 28 - runClustering set to true on top of an OLAP database (1/2)

In the case that the Boolean variable of runClustering at the FuzzyKmeansDriver is set to true, then it is time for one last map job to run, in order for the algorithm to show where and how the data points have been set to each cluster. In that case, there are not created any new database tables, and we proceed to the last map job, where again we have multiple threads doing the same job in parallel, with different data each.

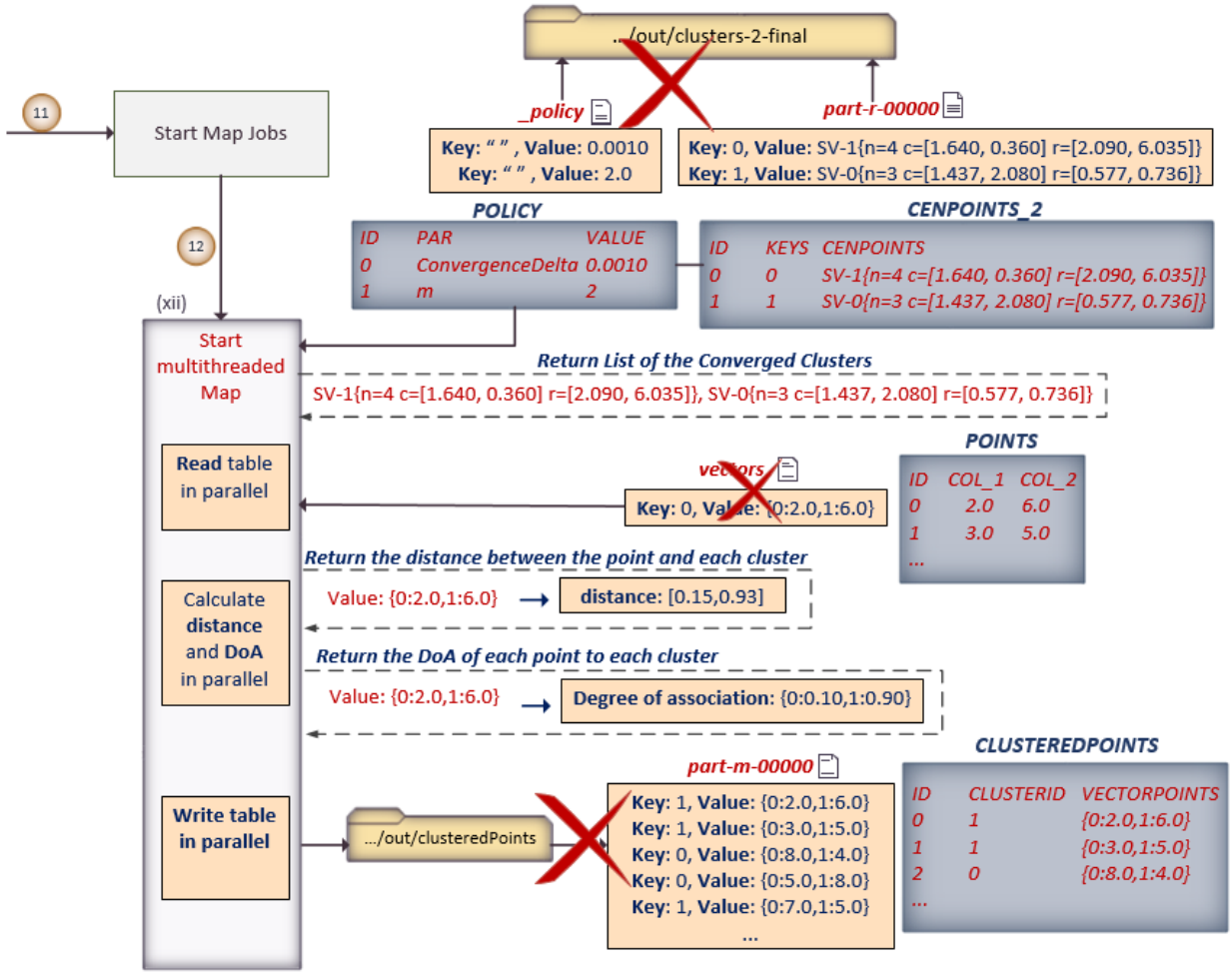

Figure 29 - runClustering set to true on top of an OLAP database (2/2)

Then, the final Map phase takes place. Each thread, reads from the database table which contains the policy variables and the database table of the converged clusters, and returns their values as a list.

Example: It reads the table *POLICY*, which is the following:

ID	PAR	VALUE
0	ConvergenceDelta	0.0010
1	m	2

Afterwards, it reads the table *CENPOINTS_2* and it returns the following list, with the values of the <key, value> pairs, of the clusters:

SV-1{n=4 c=[1.640, 0.360] r=[2.090, 6.035]}, SV-0{n=3 c=[1.437, 2.080] r=[0.577, 0.736]}

After that, each thread reads from the database table which contains all the data points, the data points that have been assigned to it, according to their identification number.
Example: Each thread reads the data points of the table POINTS, according to their assigned IDs. For instance, the first thread (thread-1) reads the first 1000 rows of the table POINTS, which is the following:

ID	COL_1	COL_2
0	2.0	6.0
1	3.0	5.0
...		

Then, each thread for each assigned to it data point calculates the distance between the point and each of the cluster centroids and it stores it in a list.
Example: The first thread (thread-1), uses the Euclidean distance measure and it calculates the Euclidean distance between the point with value {0:1.0,1:1.0} and the cluster centroids [1.640, 0.360] and [1.437, 2.080].
The list that it returns is the following:

[0.9050, 1.1650]

The same happens for all the other points, for each different thread.

Afterwards, having calculated the aforementioned distances, each thread makes use of the following formula in order to calculate the degree of association of each assigned to it point, to each cluster, and it stores it in vector format.

$$u_i = \frac{1}{\left(\frac{d_1}{d_1}\right)^{\frac{2}{m-1}} + \left(\frac{d_1}{d_2}\right)^{\frac{2}{m-1}} + \ldots + \left(\frac{d_1}{d_k}\right)^{\frac{2}{m-1}}}$$

Finally, each thread assigns the data point to the cluster with the biggest degree of association, storing it in a database table in <key, value> pairs, where the key is the cluster identifier and the value is the vectorised format of the data point.
Example: The first thread (thread-1), calculates the degree of association of the point with value {0:1.0, 1:1.0} and the clusters with the following centroids [1.640, 0.360] and [1.437, 2.080].

The vector that it returns is the following:

{0:0.6236, 1:0.3763}

As we can see, the point has a bigger degree of association with the cluster with identifier equal to 0, so it is assigned that it belong to this cluster, creating the following <key, value> pair:

key: 0, Value: {0:1.0,1:1.0}

So, the table that it creates is the *CLUSTERED_POINTS* table which contains an identification number, the cluster ID where each data point belongs to (key), accompanied with each vectorised format (value). The table is the following:

ID	*CLUSTERID*	*VECTORPOINTS*
0	*1*	*{0:2.0,1:6.0}*
1	*1*	*{0:3.0,1:5.0}*
2	*0*	*{0:8.0,1:4.0}*
…		

In that point, the description of the two different implementations finishes, and it is time for testing, using various datasets and configuring the parameters that each implementation takes as an input.

Chapter 6

Evaluation

Summary: *"In this chapter, the tests of the aforementioned implementations are taking place. More specifically, there is given a data set, with various data points, with which we execute the two different implementations in order to compare them, and understand their behavior with data. However, before all that, the system specifications and the tools that were used for the tests are described thoroughly."*

6.1 System specifications and tools

Before running the tests with the various data sets for the two implementations that were mentioned on *Chapter 5*, we should first define the system specifications, in which the tests were run.

In that case, the tests run on a computer with the following specifications (Table 6):

Specifications	Values
Operating System	Ubuntu 14.04 LTS
Memory	11.7 GiB
Processor	Intel® Core ™ i7 CPU 930 @ 2.80 GHz x 8
Graphics	GeForce GTS 250/PCIe/SSE2
OS type	64-bit
Disk	971,7 GB

Table 6 - System specifications

As for the tools that were used, these are the following:

- The *Hadoop* environment was the *1.2.1 version* (single-node Hadoop installation).
- For the tests on top of an OLAP database, as our first goal was to check whether the clustering results were correct or not, running with smaller data sets, we used *Apache Derby*, an open source relational database implemented entirely in Java.
- The *Mahout* framework was the *0.9 version.*
- The *Maven* build automation tool that was used was the *3.0 version.*
- Both implementations were written and executed in *Java* (version 7).
- The IDE that was used for the implementations was *NetBeans* (version 7.2).

6.2 Data set description

To work with the implementations, there was used a randomly created data set, which contains information about the habits of a town's citizens, about if they use or not public transports (named as *ptusages*).

In general, the data set consists of 13 different columns, which describe the contents of Table 7 and can have values between a specific range, according to what they represent. In more details, the columns of the data set are the following:

Public Transport Usage data set													
Description	Identification Number	Gender	Age	Married	Children	Educational Level	Work	Income	Driver License	Public Transport users	How often transport	Monthly Transport card holders	Which transport is most commonly used
Range of values	*1 - …*	*1/ 2*	*25 - 65*	*1/ 2*	*1/ 2/ 3/ 4/ 5*	*1/ 2/ 3/ 4/ 5/ 6/ 7*	*1/ 2*	*1 - 100000*	*1/ 2*	*1/ 2*	*1/ 2 /3 /4 /5*	*1/ 2*	*1/ 2 /3/ 4*

Table 7 - Public transport usage data set

The values in the data set are *numerical* and *separated* with commas "**,**", and represent the following:

- *Identification Number*: From 1 - ∞, as it does not have a specific range of values
- *Gender*: 1 (male) or 2 (female)
- *Age*: between 25 and 65
- *Married*: 1 (no) or 2(yes)
- *Children*: 1 (zero) or 2 (one) or 3 (two) or 4(three) or 5 (4 or more)
- *Educational Level*: 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*: 1 (no) or 2(yes)
- *Income*: between 1 and 100000 euros/year
- *Driver License*: 1 (no) or 2(yes)
- *Public Transport Users*: 1 (no) or 2(yes)
- *Public Transport Usage*: 1 (zero times) or 2(1-2 times/week) or 3 (3-4 times/week) or 4(5-6 times/week) or 5 (everyday)
- *Public Transport Card Owners*: 1 (no) or 2(yes)
- *Most Commonly used Public Transport*: 1 (bus) or 2(metro) or 3 (tram) or 4(all of them)

6.3 Tests with fuzzy kMeans

1st data set test

The first (1st) data set that was used, contained *1000 different data points*, where the values of each row were randomly created, according to the characteristics that were described earlier. A small sample of the data set can be seen below:

1,2,58,2,4,4,2,61320,2,1,5,1,1
2,1,40,1,3,5,2,20393,1,2,3,2,1
3,2,69,2,2,2,1,80449,2,1,5,2,1
4,2,44,1,3,4,2,76764,2,1,4,1,3
5,1,22,1,4,1,1,49640,1,2,3,1,3
6,1,38,1,3,5,1,89353,2,1,1,2,2
7,2,51,2,2,5,2,45200,2,2,1,1,2
8,2,27,2,3,1,1,41536,1,1,2,2,2
9,2,49,1,5,7,1,27441,2,2,4,1,4
10,1,24,2,4,4,2,27712,1,1,5,1,2
…

Having the data set in our hands, it is time to make the tests with the two different implementations.

- *Fuzzy kMeans on top of HDFS*

The parameters used in that scenario, were the following:

Parameters used for Fuzzy kMeans on top of HDFS	
Number of clusters (k): 5	
Distance Measure: Euclidean Distance	
Input: ptusages_1000	m: 2
Clustersin: part-00000	runClustering: true
Output: out	emitMostLikely: true
convergenceDelta: 0.005	threshold: 0.0
maxIterations: 20	runsequential: false

The size of the file that was stored in the HDFS was about 100 kBytes. For that reason it was only partitioned to one block.

After running the tests with these parameters, if we browse the HDFS file system, we can see that the file *ptusages_1000*, has been successfully stored to the HDFS, and there have been created the "in" and the "out" folders (Figure 30).

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
in	dir				2015-02-05 17:09	rwxr-xr-x	thanos	supergroup
out	dir				2015-02-05 17:24	rwxr-xr-x	thanos	supergroup
ptusages_1000	file	32.01 KB	1	64 MB	2015-02-05 17:09	rw-r--r--	thanos	supergroup

Figure 30 - HDFS folders and files

The "in" folder, contains the data points in vectorised form (vecpoints), while it also contains the initial centroids that have been chosen for the clusters (part-00000) (Figure 31).

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--	thanos	supergroup
vecpoints	file	120.37 KB	1	64 MB	2015-02-06 18:43	rw-r--r--	thanos	supergroup

Figure 31 - HDFS "in" folder

The "out" folder contains the various clusters folders, created in each iteration (Figure 32).

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_policy	file	0.2 KB	1	64 MB	2015-02-06 16:41	rw-r--r--	thanos	supergroup
clusteredPoints	dir				2015-02-06 16:41	rwxr-xr-x	thanos	supergroup
clusters-0	dir				2015-02-06 16:35	rwxr-xr-x	thanos	supergroup
clusters-1	dir				2015-02-06 16:35	rwxr-xr-x	thanos	supergroup
clusters-10	dir				2015-02-06 16:38	rwxr-xr-x	thanos	supergroup
clusters-11	dir				2015-02-06 16:38	rwxr-xr-x	thanos	supergroup
clusters-12	dir				2015-02-06 16:38	rwxr-xr-x	thanos	supergroup
clusters-13	dir				2015-02-06 16:39	rwxr-xr-x	thanos	supergroup
clusters-14	dir				2015-02-06 16:39	rwxr-xr-x	thanos	supergroup
clusters-15	dir				2015-02-06 16:39	rwxr-xr-x	thanos	supergroup
clusters-16	dir				2015-02-06 16:40	rwxr-xr-x	thanos	supergroup
clusters-17	dir				2015-02-06 16:40	rwxr-xr-x	thanos	supergroup
clusters-18	dir				2015-02-06 16:40	rwxr-xr-x	thanos	supergroup
clusters-19	dir				2015-02-06 16:40	rwxr-xr-x	thanos	supergroup
clusters-2	dir				2015-02-06 16:36	rwxr-xr-x	thanos	supergroup
clusters-20-final	dir				2015-02-06 16:41	rwxr-xr-x	thanos	supergroup
clusters-3	dir				2015-02-06 16:36	rwxr-xr-x	thanos	supergroup
clusters-4	dir				2015-02-06 16:36	rwxr-xr-x	thanos	supergroup
clusters-5	dir				2015-02-06 16:36	rwxr-xr-x	thanos	supergroup
clusters-6	dir				2015-02-06 16:37	rwxr-xr-x	thanos	supergroup
clusters-7	dir				2015-02-06 16:37	rwxr-xr-x	thanos	supergroup
clusters-8	dir				2015-02-06 16:37	rwxr-xr-x	thanos	supergroup
clusters-9	dir				2015-02-06 16:38	rwxr-xr-x	thanos	supergroup

Figure 32 - HDFS "out" folder

Finally, by opening a random folder among them, we can see that it contains the result sequence files of the specific iteration, including the policy files, the logs and the part-r-0000 files which contain the cluster centroids of that specific iteration (Figure 33).

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 KB	1	64 MB	2015-02-06 16:35	rw-r--r--	thanos	supergroup
_logs	dir				2015-02-06 16:35	rwxr-xr-x	thanos	supergroup
_policy	file	0.2 KB	1	64 MB	2015-02-06 16:35	rw-r--r--	thanos	supergroup
part-r-00000	file	2.96 KB	1	64 MB	2015-02-06 16:35	rw-r--r--	thanos	supergroup

Figure 33 - Results of the iteration

The results of the test can be seen in Figure 34:

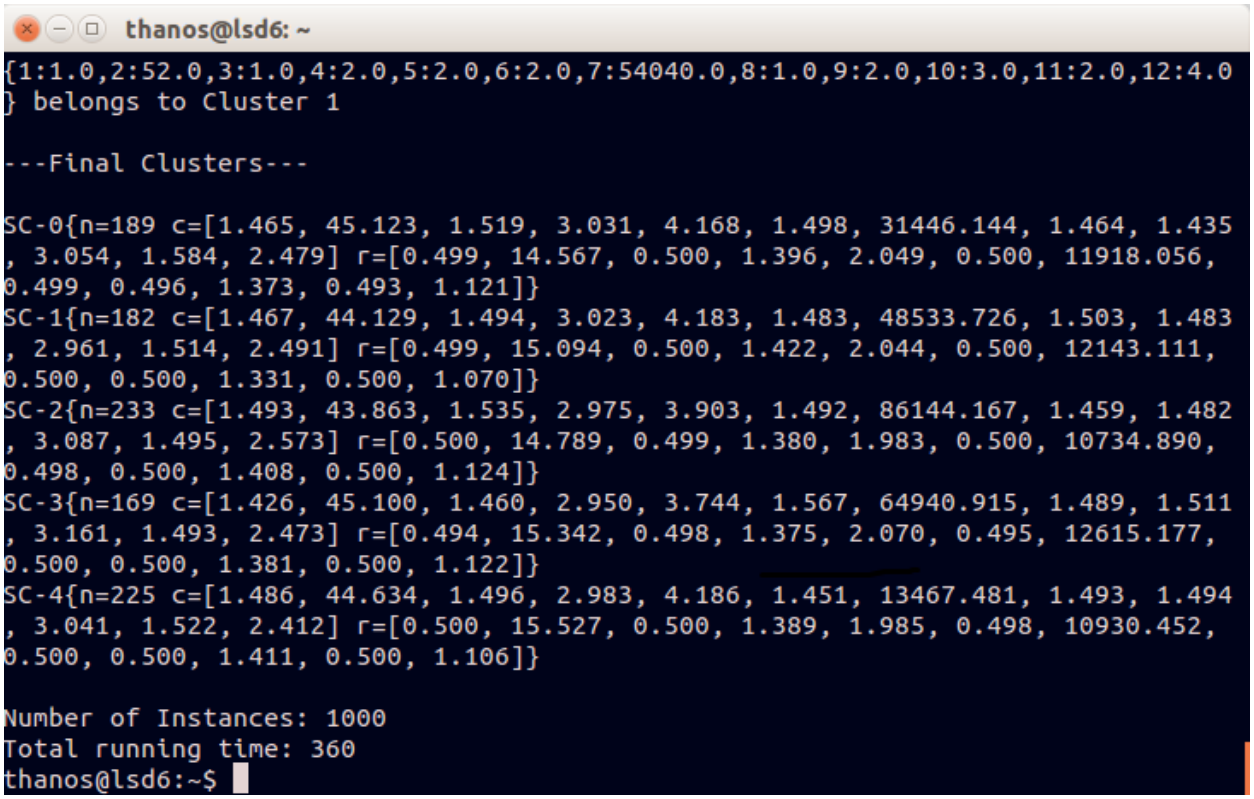

Figure 34 - Results of the 1st test (k=5)

The results can be better seen and interpreted in Table 8:

Results of the 1st test
Total time of execution: 360 sec
Total iterations: 20
Converged: False
Final Clusters: SC-4{n=225 c=[1.486, 44.634, 1.496, 2.983, 4.186, 1.451, 13467.481, 1.493, 1.494, 3.041, 1.522, 2.412] r=[0.500, 15.527, 0.500, 1.389, 1.985, 0.498, 10930.452, 0.500, 0.500, 1.411, 0.500, 1.106]} SC-3{n=169 c=[1.426, 45.100, 1.460, 2.950, 3.744, 1.567, 64940.915, 1.489, 1.511, 3.161, 1.493, 2.473] r=[0.494, 15.342, 0.498, 1.375, 2.070, 0.495, 12615.177, 0.500, 0.500, 1.381, 0.500, 1.122]} SC-2{n=233 c=[1.493, 43.863, 1.535, 2.975, 3.903, 1.492, 86144.167, 1.459, 1.482, 3.087, 1.495, 2.573] r=[0.500, 14.789, 0.499, 1.380, 1.983, 0.500, 10734.890, 0.498, 0.500, 1.408, 0.500, 1.124]} SC-1{n=182 c=[1.467, 44.129, 1.494, 3.023, 4.183, 1.483, 48533.726, 1.503, 1.483, 2.961, 1.514, 2.491] r=[0.499, 15.094, 0.500, 1.422, 2.044, 0.500, 12143.111, 0.500, 0.500, 1.331, 0.500, 1.070]} SC-0{n=189 c=[1.465, 45.123, 1.519, 3.031, 4.168, 1.498, 31446.144, 1.464, 1.435, 3.054, 1.584, 2.479] r=[0.499, 14.567, 0.500, 1.396, 2.049, 0.500, 11918.056, 0.499, 0.496, 1.373, 0.493, 1.121]}

Table 8 - Results of the 1st test (k=5)

- *Fuzzy kMeans on top of Derby*

The parameters used in that scenario, were the following:

Parameters used for Fuzzy kMeans on top of Derby	
Number of clusters (k): 5	
Distance Measure: Euclidean distance	
connection: database connect	emitMostLikely: true
convergenceDelta: 0.005	threshold: 0.0
maxIterations: 20	Numberofpoints: 1000
m: 2	k: 5
runClustering: true	Points/Thread: 100

The results of the test can be seen in Figure 35:

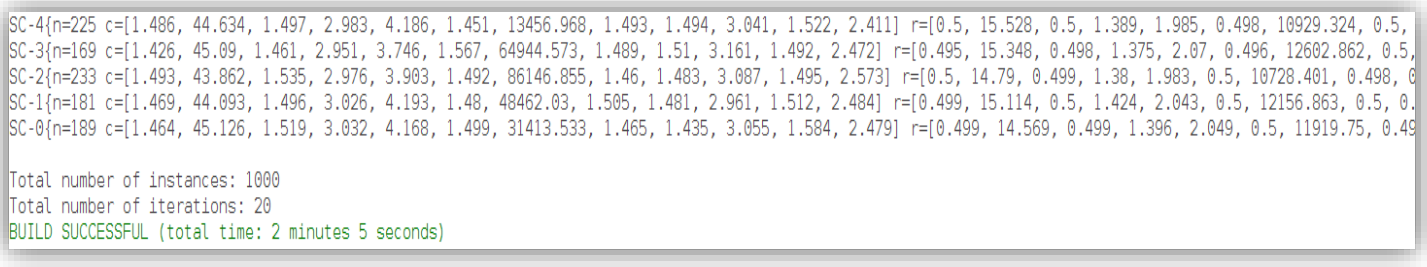

Figure 35 - Results of the 1st test (k=5)

The results can be better seen and interpreted in Table 9:

Results of the 1st test
Total time of execution: 125 sec
Total iterations: 20
Converged: False
Final Clusters: SC-4{n=225 c=[1.486, 44.634, 1.497, 2.983, 4.186, 1.451, 13456.968, 1.493, 1.494, 3.041, 1.522, 2.411] r=[0.5, 15.528, 0.5, 1.389, 1.985, 0.498, 10929.324, 0.5, 0.5, 1.411, 0.5, 1.106]} SC-3{n=169 c=[1.426, 45.09, 1.461, 2.951, 3.746, 1.567, 64944.573, 1.489, 1.51, 3.161, 1.492, 2.472] r=[0.495, 15.348, 0.498, 1.375, 2.07, 0.496, 12602.862, 0.5, 0.5, 1.382, 0.5, 1.121]} SC-2{n=233 c=[1.493, 43.862, 1.535, 2.976, 3.903, 1.492, 86146.855, 1.46, 1.483, 3.087, 1.495, 2.573] r=[0.5, 14.79, 0.499, 1.38, 1.983, 0.5, 10728.401, 0.498, 0.5, 1.408, 0.5, 1.124]} SC-1{n=181 c=[1.469, 44.093, 1.496, 3.026, 4.193, 1.48, 48462.03, 1.505, 1.481, 2.961, 1.512, 2.484] r=[0.499, 15.114, 0.5, 1.424, 2.043, 0.5, 12156.863, 0.5, 0.499, 1.334, 0.5, 1.067]} SC-0{n=189 c=[1.464, 45.126, 1.519, 3.032, 4.168, 1.499, 31413.533, 1.465, 1.435, 3.055, 1.584, 2.479] r=[0.499, 14.569, 0.499, 1.396, 2.049, 0.5, 11919.75, 0.499, 0.496, 1.373, 0.493, 1.121]}

Table 9 - Results of the 1st test (k=5)

- *Fuzzy kMeans on top of HDFS*

The parameters used in that scenario, were the following:

Parameters used for Fuzzy kMeans on top of HDFS	
Number of clusters (k): 20	
Distance Measure: Euclidean Distance	
Input: public transport (1000)	m: 2
Clustersin: part-00000	runClustering: true
Output: out	emitMostLikely: true
convergenceDelta: 0.005	threshold: 0.0
maxIterations: 20	runsequential: false

The size of the file that was stored in the HDFS was about 100 kBytes. For that reason it was only partitioned to one block.

The results can be better seen and interpreted in Table 10:

Results of the 1st test
Total time of execution: 384 sec
Total iterations: 20
Converged: False
Final Clusters (sample): … SC-14{n=48 c=[1.447, 47.990, 1.610, 3.279, 3.904, 1.522, 29083.302, 1.443, 1.432, 3.088, 1.547, 2.582] r=[0.497, 14.300, 0.488, 1.356, 2.170, 0.500, 5474.661, 0.497, 0.495, 1.296, 0.498, 1.098]} SC-15{n=61 c=[1.459, 44.678, 1.468, 3.027, 4.117, 1.403, 15789.857, 1.424, 1.544, 3.085, 1.483, 2.441] r=[0.498, 16.448, 0.499, 1.279, 2.010, 0.491, 4869.394, 0.494, 0.498, 1.395, 0.500, 1.144]} SC-16{n=38 c=[1.397, 44.745, 1.601, 2.975, 4.109, 1.452, 75378.195, 1.449, 1.531, 3.404, 1.411, 2.334] r=[0.489, 15.861, 0.490, 1.239, 1.805, 0.498, 6132.535, 0.497, 0.499, 1.406, 0.492, 1.162]} SC-17{n=51 c=[1.406, 43.520, 1.470, 2.970, 3.965, 1.588, 51480.500, 1.557, 1.495, 2.992, 1.477, 2.653] r=[0.491, 15.389, 0.499, 1.416, 2.071, 0.492, 5360.318, 0.497, 0.500, 1.317, 0.499, 1.096]} SC-18{n=46 c=[1.503, 42.484, 1.524, 2.929, 4.223, 1.350, 42083.719, 1.471, 1.451, 2.884, 1.589, 2.437] r=[0.500, 14.295, 0.499, 1.380, 2.032, 0.477, 5623.872, 0.499, 0.498, 1.269, 0.492, 1.052]} SC-19{n=44 c=[1.442, 44.079, 1.582, 2.501, 4.070, 1.376, 20358.498, 1.600, 1.397, 3.142, 1.558, 2.463] r=[0.497, 15.469, 0.493, 1.346, 1.897, 0.484, 5714.087, 0.490, 0.489, 1.522, 0.497, 1.131]}

Table 10 - Results of the 1st test (k=20)

- *Fuzzy kMeans on top of Derby*

The parameters used in that scenario, were the following:

Parameters used for Fuzzy kMeans on top of Derby	
Number of clusters (k): 20	
Distance Measure: Euclidean Distance	
connection: database connect	emitMostLikely: true
convergenceDelta: 0.005	threshold: 0.0
maxIterations: 20	Numberofpoints: 1000
m: 2	k: 20
runClustering: true	Points/Thread: 100

The results can be better seen and interpreted in Table 11:

Results of the 1st test
Total time of execution: 454 sec
Total iterations: 20
Converged: False
Final Clusters (sample): … SC-14{n=48 c=[1.447, 47.98, 1.609, 3.281, 3.903, 1.522, 29072.48, 1.443, 1.432, 3.089, 1.547, 2.581] r=[0.497, 14.301, 0.488, 1.356, 2.17, 0.499, 5469.597, 0.497, 0.495, 1.296, 0.498, 1.098]} SC-15{n=61 c=[1.459, 44.68, 1.468, 3.027, 4.117, 1.403, 15784.991, 1.424, 1.544, 3.085, 1.483, 2.441] r=[0.498, 16.449, 0.499, 1.279, 2.011, 0.49, 4867.114, 0.494, 0.498, 1.394, 0.5, 1.144]} SC-16{n=39 c=[1.397, 44.753, 1.6, 2.976, 4.109, 1.451, 75393.535, 1.449, 1.53, 3.405, 1.411, 2.335] r=[0.489, 15.864, 0.49, 1.239, 1.805, 0.498, 6126.759, 0.497, 0.499, 1.406, 0.492, 1.162]} SC-17{n=51 c=[1.407, 43.505, 1.471, 2.972, 3.972, 1.587, 51471.746, 1.558, 1.494, 2.991, 1.476, 2.65] r=[0.491, 15.404, 0.499, 1.418, 2.072, 0.492, 5347.636, 0.497, 0.5, 1.319, 0.5, 1.095]} SC-18{n=46 c=[1.503, 42.465, 1.525, 2.926, 4.222, 1.35, 42065.593, 1.47, 1.451, 2.883, 1.589, 2.439] r=[0.5, 14.277, 0.499, 1.38, 2.032, 0.477, 5617.517, 0.499, 0.498, 1.268, 0.492, 1.052]} SC-19{n=44 c=[1.442, 44.078, 1.581, 2.501, 4.07, 1.377, 20346.834, 1.6, 1.397, 3.14, 1.558, 2.464] r=[0.496, 15.471, 0.493, 1.347, 1.897, 0.485, 5711.06, 0.49, 0.489, 1.522, 0.497, 1.131]}

Table 11 - Results of the 1st test (k=20)

2nd data set test

The second (2nd) data set that was used, contained *200.000 different data points*, where the values of each row were randomly created, according to the characteristics that were described in the beginning of *Chapter 6*. A small sample of the data set can be seen below:

1,1,24,2,4,4,2,27712,1,1,5,1,2
2,2,69,1,4,5,1,4188,2,1,1,2,2
3,2,47,2,4,2,2,66786,2,2,5,1,2
4,2,44,1,3,4,2,76764,2,1,4,1,3
5,1,22,1,4,1,1,49640,1,2,3,1,3
6,1,38,1,3,5,1,89353,2,1,1,2,2
7,2,51,2,2,5,2,45200,2,2,1,1,2
8,2,27,2,3,1,1,41536,1,1,2,2,2
9,2,49,1,5,7,1,27441,2,2,4,1,1
…

Having the data set in our hands, it is time to make the tests with the two different implementations.

- *Fuzzy kMeans on top of HDFS*

The parameters used in that scenario, were the following:

Parameters used for Fuzzy kMeans on top of HDFS	
Number of clusters (k): 5	
Distance Measure: Euclidean Distance	
Input: public transport ($2*10^5$)	m: 2
Clustersin: part-00000	runClustering: true
Output: out	emitMostLikely: true
convergenceDelta: 0.005	threshold: 0.0
maxIterations: 20	runsequential: false

The size of the file that was stored in the HDFS was 6.7 MBytes. For that reason it was only partitioned to one block.

The results can be better seen and interpreted in Table 12:

Results of the 2nd test
Total time of execution: 544 sec
Total iterations: 20
Converged: False
Final Clusters: SC-0{n=36890 c=[1.504, 43.906, 1.483, 2.984, 4.015, 1.485, 50107.192, 1.496, 1.499, 2.986, 1.497, 2.495] r=[0.500, 15.283, 0.500, 1.407, 1.991, 0.500, 12198.332, 0.500, 0.500, 1.423, 0.500, 1.119]} SC-1{n=38050 c=[1.494, 44.108, 1.499, 2.989, 4.035, 1.505, 32399.983, 1.503, 1.517, 2.970, 1.506, 2.495] r=[0.500, 15.284, 0.500, 1.408, 2.008, 0.500, 12011.038, 0.500, 0.500, 1.418, 0.500, 1.119]} SC-2{n=43870 c=[1.497, 44.224, 1.509, 2.999, 3.975, 1.498, 13704.547, 1.496, 1.506, 3.010, 1.497, 2.510] r=[0.500, 15.263, 0.500, 1.427, 2.028, 0.500, 11184.638, 0.500, 0.500, 1.407, 0.500, 1.115]} SC-3{n=43590 c=[1.504, 44.211, 1.496, 2.999, 3.970, 1.506, 86334.333, 1.501, 1.498, 2.970, 1.502, 2.479] r=[0.500, 15.221, 0.500, 1.419, 1.991, 0.500, 11219.648, 0.500, 0.500, 1.414, 0.500, 1.127]} SC-4{n=37600 c=[1.499, 44.073, 1.509, 3.019, 4.004, 1.501, 67401.272, 1.499, 1.498, 2.985, 1.498, 2.491] r=[0.500, 15.380, 0.500, 1.421, 1.983, 0.500, 12081.586, 0.500, 0.500, 1.426, 0.500, 1.116]}

Table 12 - Results of the 2nd test (k=5)

- *Fuzzy kMeans on top of Derby*

The parameters used in that scenario, were the following:

Parameters used for Fuzzy kMeans on top of Derby	
Number of clusters (k): 5	
Distance Measure: Euclidean Distance	
connection: database connect	emitMostLikely: true
convergenceDelta: 0.005	threshold: 0.0
maxIterations: 20	Numberofpoints: 200000
m: 2	k: 5
runClustering: true	Points/Thread: 1000

The results can be better seen and interpreted in Table 13:

Results of the 2nd test
Total time of execution: 473 sec
Total iterations: 20
Converged: False
Final Clusters: SC-0{n=36890 c=[1.504, 43.906, 1.483, 2.984, 4.015, 1.485, 50107.192, 1.496, 1.499, 2.986, 1.497, 2.495] r=[0.500, 15.283, 0.500, 1.407, 1.991, 0.500, 12198.332, 0.500, 0.500, 1.423, 0.500, 1.119]} SC-1{n=38050 c=[1.494, 44.108, 1.499, 2.989, 4.035, 1.505, 32399.983, 1.503, 1.517, 2.970, 1.506, 2.495] r=[0.500, 15.284, 0.500, 1.408, 2.008, 0.500, 12011.038, 0.500, 0.500, 1.418, 0.500, 1.119]} SC-2{n=43870 c=[1.497, 44.224, 1.509, 2.999, 3.975, 1.498, 13704.547, 1.496, 1.506, 3.010, 1.497, 2.510] r=[0.500, 15.263, 0.500, 1.427, 2.028, 0.500, 11184.638, 0.500, 0.500, 1.407, 0.500, 1.115]} SC-3{n=43590 c=[1.504, 44.211, 1.496, 2.999, 3.970, 1.506, 86334.333, 1.501, 1.498, 2.970, 1.502, 2.479] r=[0.500, 15.221, 0.500, 1.419, 1.991, 0.500, 11219.648, 0.500, 0.500, 1.414, 0.500, 1.127]} SC-4{n=37600 c=[1.499, 44.073, 1.509, 3.019, 4.004, 1.501, 67401.272, 1.499, 1.498, 2.985, 1.498, 2.491] r=[0.500, 15.380, 0.500, 1.421, 1.983, 0.500, 12081.586, 0.500, 0.500, 1.426, 0.500, 1.116]}

Table 13 - Results of the 2nd test (k=5)

- *Fuzzy kMeans on top of HDFS*

The parameters used in that scenario, were the following:

Parameters used for Fuzzy kMeans on top of HDFS	
Number of clusters (k): 20	
Distance Measure: Euclidean Distance	
Input: public transport (1000)	m: 2
Clustersin: part-00000	runClustering: true
Output: out	emitMostLikely: true
convergenceDelta: 0.005	threshold: 0.0
maxIterations: 20	runsequential: false

The size of the file that was stored in the HDFS was 6.7 MBytes. For that reason it was only partitioned to one block.

The results can be better seen and interpreted in Table 14:

Results of the 2nd test
Total time of execution: 1117 sec
Total iterations: 20
Converged: False
Final Clusters (sample): … SC-14{n=13300 c=[1.501, 44.215, 1.509, 3.010, 3.955, 1.513, 4379.437, 1.506, 1.502, 3.004, 1.484, 2.493] r=[0.500, 15.433, 0.500, 1.423, 2.022, 0.500, 4841.779, 0.500, 0.500, 1.390, 0.500, 1.132]} SC-15{n=9520 c=[1.504, 43.874, 1.478, 3.020, 4.032, 1.479, 52429.165, 1.494, 1.498, 2.977, 1.489, 2.455] r=[0.500, 15.007, 0.500, 1.389, 1.979, 0.500, 5724.105, 0.500, 0.500, 1.437, 0.500, 1.114]} SC-16{n=9740 c=[1.480, 44.375, 1.499, 2.910, 4.125, 1.494, 24973.675, 1.483, 1.500, 2.992, 1.494, 2.467] r=[0.500, 15.115, 0.500, 1.451, 2.024, 0.500, 5657.328, 0.500, 0.500, 1.407, 0.500, 1.104]} SC-17{n=9180 c=[1.506, 44.108, 1.499, 3.016, 4.037, 1.522, 29650.978, 1.518, 1.543, 2.968, 1.528, 2.504] r=[0.500, 15.121, 0.500, 1.384, 2.021, 0.499, 5827.210, 0.500, 0.498, 1.416, 0.499, 1.134]} SC-18{n=9270 c=[1.519, 44.550, 1.519, 3.000, 4.026, 1.514, 70397.915, 1.512, 1.506, 3.021, 1.495, 2.487] r=[0.500, 15.814, 0.500, 1.441, 1.979, 0.500, 5798.279, 0.500, 0.500, 1.451, 0.500, 1.117]} SC-19{n=10930 c=[1.508, 43.297, 1.495, 3.012, 3.973, 1.505, 89799.787, 1.507, 1.499, 2.966, 1.491, 2.453] r=[0.500, 15.028, 0.500, 1.427, 1.980, 0.500, 5341.568, 0.500, 0.500, 1.380, 0.500, 1.136]}

Table 14 - Results of the 2nd test (k=20)

- *Fuzzy kMeans on top of Derby*

The parameters used in that scenario, were the following:

Parameters used for Fuzzy kMeans on top of Derby	
Number of clusters (k): 20	
Distance Measure: Euclidean Distance	
connection: database connect	emitMostLikely: true
convergenceDelta: 0.005	threshold: 0.0
maxIterations: 20	Numberofpoints: 200000
m: 2	k: 20
runClustering: true	Points/Thread: 1000

The results can be better seen and interpreted in Table 15:

Results of the 2nd test
Total time of execution: 1268 sec
Total iterations: 20
Converged: False
Final Clusters (sample): … SC-14{n=13300 c=[1.501, 44.215, 1.509, 3.010, 3.955, 1.513, 4379.437, 1.506, 1.502, 3.004, 1.484, 2.493] r=[0.500, 15.433, 0.500, 1.423, 2.022, 0.500, 4841.779, 0.500, 0.500, 1.390, 0.500, 1.132]} SC-15{n=9520 c=[1.504, 43.874, 1.478, 3.020, 4.032, 1.479, 52429.165, 1.494, 1.498, 2.977, 1.489, 2.455] r=[0.500, 15.007, 0.500, 1.389, 1.979, 0.500, 5724.105, 0.500, 0.500, 1.437, 0.500, 1.114]} SC-16{n=9740 c=[1.480, 44.375, 1.499, 2.910, 4.125, 1.494, 24973.675, 1.483, 1.500, 2.992, 1.494, 2.467] r=[0.500, 15.115, 0.500, 1.451, 2.024, 0.500, 5657.328, 0.500, 0.500, 1.407, 0.500, 1.104]} SC-17{n=9180 c=[1.506, 44.108, 1.499, 3.016, 4.037, 1.522, 29650.978, 1.518, 1.543, 2.968, 1.528, 2.504] r=[0.500, 15.121, 0.500, 1.384, 2.021, 0.499, 5827.210, 0.500, 0.498, 1.416, 0.499, 1.134]} SC-18{n=9270 c=[1.519, 44.550, 1.519, 3.000, 4.026, 1.514, 70397.915, 1.512, 1.506, 3.021, 1.495, 2.487] r=[0.500, 15.814, 0.500, 1.441, 1.979, 0.500, 5798.279, 0.500, 0.500, 1.451, 0.500, 1.117]} SC-19{n=10930 c=[1.508, 43.297, 1.495, 3.012, 3.973, 1.505, 89799.787, 1.507, 1.499, 2.966, 1.491, 2.453] r=[0.500, 15.028, 0.500, 1.427, 1.980, 0.500, 5341.568, 0.500, 0.500, 1.380, 0.500, 1.136]}

Table 15 - Results of the 2nd test (k=20)

6.4 Evaluation of the results

After having run the different tests, we have the following results as for the two implementations:

1st data set test

- 1st Scenario

During the first test, the data set had *1000 different data points*, and the number of the chosen clusters was *k=5*.

By examining the results, we can see the following:

HDFS implementation	Derby implementation
File size: 100 kBytes	
Distance Measure: Euclidean Distance	
Clusters: 5	Clusters: 5
Total time of execution: 360 sec	Total time of execution: 125 sec
Total iterations: 20	Total iterations: 20
Converged: False	Converged: False
Differences between the final clusters: False	

Table 16 - 1st data set test - 1st scenario results

In short, someone could observe that the fuzzy kMeans implementation on top of Derby, is *2.88 times faster*, than the other implementation. As for the *final clusters*, they have exactly the *same values*, for the same number of iterations.

- 2nd Scenario

During the first test, the data set had *1000 different data points*, and the number of the chosen clusters was *k=20.*

By examining the results, we can see the following:

HDFS implementation	Derby implementation
File size: 100 kBytes	
Distance Measure: Euclidean Distance	
Clusters: 20	Clusters: 20
Total time of execution: 384 sec	Total time of execution: 454 sec
Total iterations: 20	Total iterations: 20
Converged: False	Converged: False
Differences between the final clusters: False	

Table 17 - 1st data set test - 2nd scenario results

In short, someone could observe that the fuzzy kMeans implementation on top of HDFS, is a *0.84 times faster* than the other implementation, which runs on top of Derby. As for the *final clusters*, they have exactly the *same values*, for the same number of iterations.

2nd data set test

- 1st Scenario

During the second test, the data set had *200.000 different data points*, and the number of the chosen clusters was *k=5.*

By examining the results, we can see the following:

HDFS implementation	Derby implementation
File size: 6.7 MBytes	
Distance Measure: Euclidean Distance	
Clusters: 5	Clusters: 5
Total time of execution: 544 sec	Total time of execution: 473 sec
Total iterations: 20	Total iterations: 20
Converged: False	Converged: False
Differences between the final clusters: False	

Table 18 - 2nd data set test - 1st scenario results

In short, someone could observe that the fuzzy kMeans implementation on top of Derby, is again *1.15 times faster*, than the other implementation. As for the *final clusters*, they have exactly the *same values*, for the same number of iterations.

- 2nd Scenario

During the second test, the data set had *200000 different data points*, and the number of the chosen clusters was *k=20***.**

By examining the results, we can see the following:

HDFS implementation	Derby implementation
File size: 6.7 MBytes	
Distance Measure: Euclidean Distance	
Clusters: 20	Clusters: 20
Total time of execution: 1117 sec	Total time of execution: 1268 sec
Total iterations: 20	Total iterations: 20
Converged: False	Converged: False
Differences between the final clusters: False	

Table 19 - 2nd data set test - 2nd scenario results

In short, someone could observe that the fuzzy kMeans implementation on top of HDFS, is *0.88 times faster* than the other implementation, which runs on top of Derby. As for the *final clusters*, they have exactly the *same values*, for the same number of iterations.

Test Conclusions

Having executed the aforementioned tests, we could understand that *when the number of initial clusters becomes larger, then the Fuzzy kMeans algorithm on top of Derby, converges slower.*

On the other hand, *when the number of the initial clusters gets smaller, then the Fuzzy kMeans Algorithm on top of Derby, converges faster.*

As for the final results, we can see that *both implementations have exactly the same clustering results*, running for the *same number of iterations*. For that reason, we cannot have a more clear view, on which implementation converges faster or not, as the maximum iterations had been set by default to 20, for each of the different scenarios.

Additionally, as we can view from the results, *the data points have been assigned well to the clusters,* meaning that in each cluster it belongs almost equal number of data points, *not having large distances between them.* It must be mentioned that if the maximum iterations had been set with a larger value, then we could probably have better results concerning the assignment of the data points to each cluster, due to the fact that the cluster centroids would have better final positions (or would have converged, meaning that they would no longer move from their current position).

Having all that in mind, we could clearly understand that *each implementation works better, when the number of the initial clusters gets bigger or not, as well.*

However, one should have in mind that there should be done more and more tests, on different data sets, with different parameters for each implementation, in order to have a more clear image about which implementation works better. In addition, the results and the execution time can vary a lot according to the specifications of the machine that were tested, and according to how each implementation is configured to run (e.g. Hadoop Configuration).

In the previous implementations' tests, the reason for why the tests have been done in small data sets, is because we wanted for the beginning, to be sure that it runs and outputs correct clustering results. In addition, we already know that Hadoop is designed for working with big data, but the aforementioned project that was implemented is just a prototype, and should be first tested with easily interpretable and understandable data sets.

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 an OLAP database.

As it was mentioned above, one should keep in mind that the new *fuzzy kMeans implementation*, is a *new-born project*, which means that for sure it will have to be debugged and corrected in the way that it works. However, as one could see, the final results that it produces do not differ much from the results that the other implementation produces, meaning that it should be considered as a *good effort* for a prototype project, which has been developed in the last months.

Chapter 7

Conclusions

Summary: *"In this chapter, the final conclusions after the completion of the thesis are given, accompanied with some of my future plans, which have to do with the improvement of the algorithm's implementation and the study of new technologies and data mining algorithms."*

7.1 Conclusions

After the completion of this thesis, it could be said that the overall goals which have been set in *Chapter 1*, have been satisfyingly fulfilled and accomplished.

In more details, nowadays, as we have seen data has a significant importance. The first goal of the thesis was to focus and study about the data accompanied with the era we live in. For that reason, there was an extensive state of the art study about the data in general, including the data analysis process from which someone can retrieve valuable information and meaningful insights.

Furthermore, we dealt with the fields of data preprocessing, which is a very important step of the data analysis, in order to gain the best insights. As for this sector, there were given sufficient and understandable definitions, targeting the different kinds of data that someone can face with and the different ways of "cleaning" the data from various types of noises and faults.

Afterwards, it was given significant importance to the phenomenon of the data evolution, the big data. It was analyzed what big data is and how they can be connected with our daily lives, including the value of analyzing and exporting knowledge out of it, keeping in mind their large volume, velocity and variety.

Moreover, we dealt with the fields of the knowledge discovery in databases (KDD) and data mining, trying to analyze with the most efficient and in-deep details way the techniques, the methodologies and the ways with which data mining is accomplished.

Last but not least, there was given a detailed literature review about clustering, the data mining technique, which is used every day to gain results, valuable information and insights out of unknown and unlabeled data. Explanations were given about what is clustering, which are the most well-known clustering techniques, clustering algorithms and in general where and how it is used.

It should be mentioned, that there were given more details to that part, as the main goal of the thesis was to understand the clustering algorithms and techniques, and implement a clustering algorithm, working on top of an OLAP database.

Having explained the state of the art on data analysis, it was time to study about Hadoop and Mahout, in order to be able to transfer the way that Mahout performs clustering with Hadoop, in a different environment, using an OLAP database and a different distributed processing concept.

As for Hadoop, there was given an in deep details literature review about what in general is Hadoop. We discovered that Hadoop is a distributed processing framework, which consists of various distributed components and a complicated architecture. We studied about the different Hadoop cluster components, and a detailed explanation was given about its architecture.

Additionally, we studied in details about the Hadoop's distributed file system (HDFS) and the MapReduce framework, which are the basic components of its ecosystem, and which were important to understand, in order to continue with the algorithm's implementation chapters.

Afterwards, it was time to study about a data mining software which is included in the apache Hadoop's ecosystem, Mahout. Mahout was very important to understand, as the whole algorithm's implementation was running using the machine learning libraries that Mahout provides, so it was given significant importance on what is Mahout, how it works and how can someone perform clustering techniques with it.

Having in mind all of the aforementioned, it was time for the implementation. The clustering algorithm that was chosen was the fuzzy kMeans algorithm, which belongs to the family of the fuzzy clustering algorithms. A detailed description of how it works in general was given, and after that there was mentioned a deep-detailed explanation of how it works on top of HDFS, using the MapReduce paradigm and how it works with its new implementation, running on top of an OLAP database, using a distributed <key, value> pair concept.

Following, it was time to run some tests, with the two aforementioned algorithm implementations. A short description of the tools and the system specifications were given. Afterwards, the data set used for the tests was explained and finally the results of the various tests were written, followed by a brief evaluation and some additional comments.

Generally, we are able to see that the use of data occupies more and more the technology sector. Every day, the data that is transferred among the companies, the organizations, the firms and between individuals, is multiplied in a very large scale,

creating difficulties as for the separation between important and non-important information. As it becomes comprehensible, the data mining field is used in order to sort, to classify and to exploit this data, so to either extract valuable insights or prepare the data in such a way, in order to give back the biggest benefit to those who need it.

Moreover, we are able to see that data have to be analyzed, processed and evaluated in a faster and more efficient way, as it becomes more and more complicated. For that reason, there must always be various implementations of algorithms, which will try to work with data in a more efficient and effective way. These algorithms should use more and more the distributed systems framework, in order to work and have results in a parallel way, but they also have to be evaluated, corrected and tested many times, before replacing the older implementations.

However, one should keep in mind that there should not be a fully replace of the older algorithm implementations, as in some points there must be most productive, efficient or effective, than the newer implementations, according to the data that has to be mined and processed.

There is no doubt that the era we live in, gets stormed by a very big amount of problems, which have to do with the data and its process, for which different kinds of solutions are being discovered daily. Some of these problems, are listed below:

- *Larger databases*: Databases with hundreds of fields and tables and millions of records are beginning to appear.
- *High dimensionality*: There can be a large number of fields, including a large number of records in the database. So, the dimensionality of the problem gets high.
- *Changing data and knowledge*: Data that is changing quickly can make previously discovered techniques or patterns invalid.
- *Missing and noisy data*: Due to the fact that data become larger and larger, there is the problem that they contain many noisy data, so they require a pre-processing method for cleaning it.
- *Complex relationships between fields*: The relationships between the data are becoming more and more complex, making ii very hard to identify them.

In other words, we observe that it has to be given significant importance to all the aforementioned problems, by producing more and more capable, efficient, effective, user-friendly and "low-cost" algorithms, for the various data mining techniques. However, in order to become something feasible, it is required a full comprehension of what has been previously created and carried out by scientists and researchers, so to be someone able to reengineer and improve an existing technique. It should be mentioned that the last part, requires a lot of preparation, big effort, right guidance and a huge amount of experience, in order for a complete solution to be created.

7.2 Future plans

As for the *future plans*, after the fulfillment of the thesis above, all the aforementioned that were explained and analyzed, have to be examined and studied using various kinds of data sets. As a result, we have to find larger data sets, noisy and non-noisy, with various kinds of attributes, in order to test how the implementations work with different kind of data.

Additionally, it is mandatory that the implementations be configured in order to work with both numerical and non-numerical data, as the data that we find every day, unfortunately are not only in number format.

As for the fuzzy kMeans implementation on top of an OLAP database, the preprocessing step has to be configured in order to be able to read and write the data set into a table, faster and concurrently.

Furthermore, the implementation above could be changed in order to work simultaneously with various tables, and having faster and more efficient results.

However, as most of the implementations have different kinds of bugs or errors that make them inefficient, the whole project should be evaluated by different people, in order to find and try to solve the various problems that may appear, and were not noticed until today. This is a long time procedure, which has to be done in order for the algorithm to work in the most efficient way.

As it has been stated, because of the fact that we wanted to test the implementations to check if their results were correct or not, we performed the tests running on top of a relational database, with small data sets. So one of the future plans is to test and to configure the whole implementation on top of an OLAP database, using massive amounts of data.

Afterwards, one of the future plans is to study different techniques, similar to the MapReduce paradigm, in order to discover the differences and be able to evaluate each one of them. Having all that in mind, for sure a better and more efficient implementation could be delivered, just by combining the gained knowledge from the above study.

Furthermore, as we have seen, data mining is not only clustering, and clustering is not only the fuzzy kMeans algorithm. So, more and more algorithms should be studied, including the different data mining techniques, and why not, it would be a nice idea to try to implement, reengineer or create a new data mining algorithm, which would solve many of the today's big data challenges.

The kMeans, and more specifically the fuzzy kMeans algorithm, is one of the most easy to understand algorithms, so we should not stay only to that. There are hundreds of algorithms that are waiting to be explored and implemented, using different ways of distributing, analyzing and gaining insights from the data.

Last but not least, Mahout is just a sub-project of Hadoop's ecosystem. There are many other projects similar to Mahout, with machine learning libraries which can probably be more effective, according to the data set and the algorithms that implement.

One should keep in mind that our era consists of non-static data, as they change hour by hour, day by day, so it must be given significant importance to the streaming data, in order to gain information and valuable insights, using algorithms which perform data mining techniques in real time. This could probably be a big challenge, to study, to understand, to test, to implement and to produce algorithms for real-time analysis.

However, as it was mentioned before, such things require a big effort and experience, which hopefully I am going to obtain in the next few years with the appropriate preparation, study and supervision.

References

[1] Data Analysis. Publicly available from:
http://en.wikipedia.org/wiki/Data_analysis

[2] Data Analytics. Publicly available from:
http://searchdatamanagement.techtarget.com/definition/data-analytics

[3] Making Sense of Data I: A Practical Guide to Exploratory Data Analysis and Data Mining, 2nd Edition, by Glenn J. Myatt; Wayne P. Johnson, Published by John Wiley & Sons, 2014

[4] Practical Data Analysis by Hector Cuesta, Published by Packt Publishing, 2013

[5] Data Analysis with Open Source Tools by Philipp K. Janert, Published by O'Reilly Media, Inc., 2010

[6] Big Data and Analytics: Seeking Foundations for Effective Privacy Guidance. Publicly available from:
http://www.hunton.com/files/Uploads/Documents/News_files/Big_Data_and_Analytics_February_2013.pdf

[7] Planning for Big Data by Edd Dumbill, Published by O'Reilly Media, Inc., 2012

[8] What is Big Data. Publicly available from:
http://www.sas.com/en_us/insights/big-data/what-is-big-data.html

[9] Big Data Explained. Publicly available from:
http://www.mongodb.com/big-data-explained

[10] Big Data. Publicly available from:
http://www.webopedia.com/TERM/B/big_data.html

[11] Big Data Analytics. Publicly available from:
http://www.sas.com/en_us/insights/analytics/big-data-analytics.html

[12] Big Data Analytics. Publicly available from:
http://www.webopedia.com/TERM/B/big_data_analytics.html

[13] Big Data Analytics: Turning Big Data into Big Money by Frank J. Ohlhorst, Published by John Wiley & Sons, 2012

[14] Business Intelligence. Publicly available from:
http://en.wikipedia.org/wiki/Business_intelligence

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

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

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

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

[19] Knowledge Discovery in Databases. Publicly available from:
https://www.udemy.com/blog/knowledge-discovery-in-databases/

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

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

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

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

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

[25] An Exploration of Classification Prediction Techniques in Data Mining: The insurance domain by Samuel Odei Danso, Bournemouth University, September, 2006

[26] Predictive Analytics with Data Mining: How it works? Publicly available from: http://www.predictionimpact.com/predictive.analytics.html

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

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

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

[30] Aggregating Multiple Instances in Relational Database using Semi-Supervised Genetic Algorithm-based Clustering Technique, Rayner Alfred, Dimitar Kazakovi, Computer Science Department, York University, England

[31] A Comprehensive Overview of Basic Clustering Algorithms, Glenn Fung, June 22, 2001

[32] Data Clustering Algorithms. Publicly available from: https://sites.google.com/site/dataclusteringalgorithms/

[33] Cluster Analysis. Publicly available from: http://en.wikipedia.org/wiki/Cluster_analysis

[34] Data Clustering Techniques Qualifying Oral Examination Paper, Periklis Andritsos, Department of Computer Science, University of Toronto, March 11, 2002

[35] An Overview on Clustering Methods, T. Soni Madhulatha, Alluri Institute of Management Sciences, Warangal, IOSR Journal of Engineering, Vol. 2(4) pp: 719-725, April 2012

[36] Density-Based, Grid-Based and Model-Based Clustering. Publicly available from: http://www.pierlucalanzi.net/wp-content/teaching/dmtm/DMTM0809-09-ClusteringDensityModel.pdf

[37] Image Processing and Pattern Recognition: Fundamentals and Techniques, by Frank Y. Shih, May 3, 2010

[38] Hadoop For Dummies®, Special Edition. Published by. John Wiley & Sons Canada

[39] Pro Hadoop by Jason Venner, Published by Apress, 2009

[40] Hadoop Operations and Cluster Management Cookbook by Shumin Guo, Published by Packt Publishing, 2013

[41] Hadoop in Action by Chuck Lam, Published by Manning Publications, 2010

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

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

[44] Understanding Hadoop Ecosystem. Publicly available from: http://docs.hortonworks.com/HDPDocuments/HDP1/HDP-1.2.4/bk_getting-started-guide/content/ch_hdp1_getting_started_chp2.html

[45] Hadoop Advantages and Disadvantages. Publicly available from: http://www.j2eebrain.com/java-J2ee-hadoop-advantages-and-disadvantages.html

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

[47] Hadoop Cluster Architecture and Core components. Publicly available from: http://saphanatutorial.com/hadoop-cluster-architecture-and-core-components/

[48] Hadoop Core components. Publicly available from: http://blog.aziksa.com/2013/07/hadoop-core-components/

[49] What is Apache Hadoop? Publicly available from: http://hortonworks.com/hadoop/

[50] Addressing NameNode Scalability Issue in Hadoop Distributed File System using Cache Approach, by Debajyoti Mukhopadhyay, Chetan Agrawal, Devesh Maru, Pooja Yedale, Pranav Gadekar, Department of Information Technology Maharashtra Institute of Technology Pune, India

[51] Hadoop Operations Basic. Publicly available from: http://www.slideshare.net/HafizurRahman4/hadoop-operations-basic

[52] Apache Pig. Publicly available from: http://pig.apache.org/

[53] Data Organization. Publicly available from: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Organization

[54] What is the Hadoop Distributed File System (HDFS). Publicly available from: http://www-01.ibm.com/software/data/infosphere/hadoop/hdfs/

[55] Hadoop Distributed File System (HDFS). Publicly available from: http://hortonworks.com/hadoop/hdfs/

[56] An introduction to Apache Hadoop for Big Data: http://opensource.com/life/14/8/intro-apache-hadoop-big-data

[57] Hadoop Real-World Solutions Cookbook by Jon Lentz; Brian Femiano; Jonathan R. Owens, Published by Packt Publishing, 2013

[58] Hadoop: The Definitive Guide, 3rd Edition by Tom White, Published by O'Reilly Media, Inc., 2012

[59] Hadoop for Beginners guide. Publicly available from: http://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/Hadoop-Beginners guide.pdf

[60] Mining of Massive Datasets by Jeffrey David Ullman; Anand Rajaraman, Published by Cambridge University Press, 2011

[61] Hadoop in Action by Chuck Lam, Published by Manning Publications, 2010

[62] Distributed Systems in small scale research environments: Hadoop and the EM algorithm, by Jason Remington, Department of Computer Science, Colorado State University, 2011

[63] MapReduce Tutorial. Publicly available from: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[64] Hadoop MapReduce. Publicly available from: ibm.com/support/knowledgecenter/SSPT3X_2.1.2/com.ibm.swg.im.infosphere.biginsights.product.doc/doc/c0057842.html

[65] Hadoop V1 Architecture. Publicly available from: http://mikepluta.com/tag/tasktracker/

[66] Map-Reduce. Publicly available from: http://docs.mongodb.org/manual/core/map-reduce/

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

[68] Apache Mahout: Scalable machine learning for everyone. Publicly available from: http://www.ibm.com/developerworks/library/j-mahout-scaling/

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

[70] Apache Mahout, Hadoop's original machine learning project is moving on from MapReduce. Publicly available from: https://gigaom.com/2014/03/27/apache-mahout-hadoops-original-machine-learning-project-is-moving-on-from-mapreduce/

[71] Fuzzy K-Means. Publicly available from: http://mahout.apache.org/users/clustering/fuzzy-k-means.html

[72] What is the difference between K-Means and Fuzzy-C Means Clustering? Publicly available from: http://www.quora.com/What-is-the-difference-between-K-Means-and-Fuzzy-C-Means-Clustering

[73] Fuzzy Clustering. Publicly available from: http://en.wikipedia.org/wiki/Fuzzy_clustering

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

[75] Mahout: Fuzzy k-Means Clustering. Publicly available from: http://ylzhj02.iteye.com/blog/2078695

[76] Fuzzy k-Means. Publicly available from: http://mahout.apache.org/users/clustering/fuzzy-k-means.html

[77] Gene Expression Clustering, by Franck Dernoncourt (2012), Arvind Thiagarajan (2011), Tahin Syed (2010), Barrett Steinberg and Brianna Petrone (2009), Mia Y. Qia (2008) October 30, 2012