ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ

Αναγνώριση Υποχώρων Δεδομένων βασιζόμενη στα ερωτήματα

Φουντάς Παναγιώτης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων
Δρ. Κολομβάτσος Κωνσταντίνος

Λαμία 2022

UNIVERSITY OF THESSALY

SCHOOL OF SCIENCE

INFORMATICS AND COMPUTATIONAL BIOMEDICINE

Query Driven Data Subspace Mapping

Fountas Panagiotis

MASTER THESIS
Supervisor
Dr. Kolomvatsos Konstantinos

Lamia 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ

ΚΑΤΕΥΘΥΝΣΗ
«ΠΛΗΡΟΦΟΡΙΚΗ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΑΣΦΑΛΕΙΑ, ΔΙΑΧΕΙΡΙΣΗ ΜΕΓΑΛΟΥ ΟΓΚΟΥ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ»

Αναγνώριση Υποχώρων Δεδομένων βασιζόμενη στα ερωτήματα

Φουντάς Παναγιώτης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων
Δρ. Κολομβάτσος Κωνσταντίνος

Λαμία 2022

«Υπεύθυνη Δήλωση μη λογοκλοπής και ανάληψης προσωπικής ευθύνης»

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, και γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα και ενυπογράφως ότι η παρούσα εργασία με τίτλο [«Αναγνώριση Υποχώρων Δεδομένων βασιζόμενη στα ερωτήματα»] αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές από τις οποίες χρησιμοποίησα δεδομένα, ιδέες, φράσεις, προτάσεις ή λέξεις, είτε επακριβώς (όπως υπάρχουν στο πρωτότυπο ή μεταφρασμένες) είτε με παράφραση, έχουν δηλωθεί κατάλληλα και ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο ΔΗΛΩΝ

Ημερομηνία

Υπογραφή

Αναγνώριση Υποχώρων Δεδομένων βασιζόμενη στα ερωτήματα

Φουντάς Παναγιώτης

Τριμελής Επιτροπή:

Ονοματεπώνυμο, Κολομβάτσος Κωνσταντίνος

Ονοματεπώνυμο, Λουκόπουλος Αθανάσιος

Ονοματεπώνυμο, Τζιρίτας Νικόλαος

CONTENTS

NOTATION TABLE

Symbol	Description
$G = \{G_1, G_2, \dots, G_K\}$	The set of clusters as it is created by K-Means
g_i	Center of the Cluster G_i created by K-Means
$D = \{p_1, p_2, \dots, p_n\}$	Set of n vectors.
n	Number of vectors in the dataset
d	Number of dimensions in each vector
l	Number of iterations that the K-Means needs to cluster data
J	Objective function of K-Means
FCM	Fuzzy C-Means
KM	K-Means
u_{ik}	Membership value for the i^{th} data point in the k^{th} cluster
$C = \{C_1, C_2, C_3, \dots, C_\gamma\}$	The set of clusters as it is created by FCM
c_i	Center of the Cluster S_i created by FCM
m	Fuzzy parameter (FCM parameter)
β	Value for stopping criterion in FCM
$\mathbf{U}$	Membership matrix $c \times n$
$J_m(\mathbf{U}, \mathrm{S})$	Objective function of FCM
w	Number of Distributed Database Management Systems
$DB = \{DB_1, DB_2, \dots, DB_w\}$	A set of Distributed Database Management Systems
$Q = \{q_1, q_2, \dots, q_z\}$	A set of queries from the users
z	Number of queries that server received by the users.
$Z = \{Z_1, Z_2, \dots, Z_w\}$	Set of groups of IoT devices
SV	It symbolizes the Server of our scenario
$X^t = [x_1^t, x_2^t, \dots, x_d^t]$	The form of the multivariate vectors that are reported by a IoT device at time instance t.
t	Time instance
W	Window Size for the training phase in HCBM and HMCM
Φ	Number of data vectors that a query has to receive as answer
DBMS	Database Management System
DDBMS	Distributed Database Management System

MDBSs	Multidatabase systems
BM	Baseline Method
DDB	Distributed Database
QBL	Query-Based Learning
GCS	Global Conceptual Schema
LCS	Local Conceptual Scheme
LIS	Local Internal Scheme
ES	External Schema
PAM	Partitioning Around Medoids
BIRCH	Balanced Iterative Reducing and Clustering using Hierarchies
CF	Clustering Feature
CF-tree	Clustering Feature Tree
LS	Linear Sum
SS	Sum of Squares
N	Number of data that belong to a cluster (BIRCH algorithm)
B	Branching factor
L	Maximum number of entries in a leaf node
T	Maximum diameter of subclusters stored at a leaf node of a CF-tree
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
OPTICS	Ordering Points To Identify the Clustering Structure
Ω	Clusters in the Chameleon algorithm
minPts	the minimum number of data points inside the neighborhood of a point at DBSCAN algorithm
ε	the radius at DBSCAN algorithm
SV	Server
AOM	Area Overlap Metric

ABSTRACT

The increased adoption of various types of computer systems and smart devices in several areas has created an enormous amount of data. In parallel, the need of several applications and users for a part of these data for the execution of tasks and the extraction of knowledge has provoked the injection of tremendous number of queries per second into the servers of distributed databases. Due to this phenomenon, a significant process is the efficient response of these queries both in time requirements and the detection of appropriate data excluding unrequired data points. In this thesis, we propose an hierarchical query-driven clustering model to perform efficient data mapping for future incoming queries in distributed databases. We differentiate from the state-of-the-art solutions by involving in the same model the process of Query-Based Learning (QBL) with a hierarchical clustering and different types of clustering. The performance of the proposed model is evaluated through a variety of experimental scenarios being also exposed by numerical results.

Keywords: Data mapping, Data Management, Query-Based Learning, Hierarchical Clustering, Data Retrieval

CHAPTER 1: INTRODUCTION

In the era of the rapid increment of the production of data, analysts have to deal with a tremendous number of data which are located in different databases. For this reason, many challenges have been arisen for the management and extraction of knowledge from them such as data caching, the assurance of the quality of data by eliminating anomalies in data, missing values, data mapping etc. Data mapping is the process which gathers data from multiple datasets into a single dataset and stores them in a uniform way. This process plays a significant role in a variety of processes such as data migration, data integration, data warehousing etc.

In this thesis, we propose an hierarchical clustering model for the detection of the appropriate data for the execution of queries sent into a server in the minimum possible time. We elaborate on a model which takes into consideration the required data of a number of historical queries which arrive in the server during a time interval to detect and collect the data that a future query will require, without scanning the entire distributed databases. We focus on the use of two different types of algorithms for clustering i.e., the fuzzy clustering and the hard clustering, using the Fuzzy C-Means (FCM) and the K-Means (KM) algorithms, respectively. These clustering methods are used to create clusters with similar queries, based on the data that they need for their execution. The proposed model examines the queries that are coming into the server to find the most similar clusters upon them. Based on the relevant literature, we involve in our model a mechanism which computes the overlap of the area of interest between two queries. The novelty of this thesis is that we combine the QBL with a hierarchical clustering scheme into a model which has the ability to predict which of the data it will have to retrieve for similar future queries. The main contributions of this thesis are as follows:

- We propose an hierarchical clustering-based model combining two different types of clustering methods for the detection of the appropriate data as responses to future queries in the minimum possible time and with the minimum error.
- We adopt QBL to retrieve the appropriate data for the user's query relying on the retrieved data of previously executed queries.
- We argue on an area overlapping metric between the areas of the queries i.e., the area which is formed by the boundaries of the query to detect the existence of data points of common interest.

CHAPTER 2: RELATED WORK

The data management processes constitute an important factor for the effectiveness of various tasks and operations such as the extraction of analytics from a set of data. The research community has proposed several models and mechanisms for the improvement of various data management processes. Caching is a crucial data management process which prevents the burden of the memory with unnecessary data. In [1] the authors present a distributed framework for cost-based caching of multi-dimensional raw arrays in native format. More specifically, the authors adopt a R-tree index to eliminate extra raw files from processing and they propose cost-based algorithms for distributed cache eviction and placement, taking into consideration a historical query workload. A Caching System for Graph Queries is presented in [2]. The authors present a full-fledged caching system named GrapheCache (GC) for graph queries which deals with the resource and dynamic management of the cache index. Also, they involve a semantic graph cache in their system which leading to a speedup for its performance and a cache admission control mechanism improving the performance gains of GC. Additionally, they propose a new solution for the general subgraph isomorphism problems by GC while they refer a number of graph cache replacement strategies including a novel hybrid graph cache replacement policy. Another major data management process is the effective response in queries from users and applications. The work discussed in [3] presents BlinkDB a massively parallel, approximate query engine for the execution of interactive SQL queries in large amount of data and allows the users to trade-off query accuracy for response time. Especially, BlinkDB is based on an adaptive optimization framework and a dynamic sample selection. The former is used to build and maintain a set of multi-dimensional samples from the database over time while the dynamic sample selection strategy is used to select an appropriately sized sample based on a query's accuracy or response time requirements. In addition, the authors of [4] present a solution for the assigning of the queries and tasks in the appropriate edge computing nodes in order to reduce as much as possible the response time. For this purpose, the authors propose a method for the estimation of the computational burden that an allocation of a query will be added to a node. Also, the authors develop an ensemble similarity scheme responsible to deliver the complexity class for each query or task and a probabilistic decision-making model. Moreover, in [5], the authors develop Relational Sum Product Networks (RSPNs) i.e., a type of deep probabilistic models over databases which can capture important characteristics of a database. They propose a probabilistic query compilation approach to translate the incoming queries into probability and expectations for RSPNs. These former two methods are combined in a proposal data-driven model that can be used for different tasks such as query answering, or cardinality estimation called DeepDB. The partition of large amount of data into smaller parts has arisen several challenges. In [6], the authors define the concept of a Query Controller (QC) that assign each of the queries into a processor which is placed in front of each data partition. Based on this technology they develop a framework for query assignment which involves two learning schemes, i.e., a Reinforcement Learning (RL) and a clustering scheme. Also, they propose a multiple Q-tables scheme as knowledge base of the QC in the RL case and a technique for deriving the level of compactness of the created clusters in the clustering scheme to deliver the best possible QP for each assignment. In [7], the authors introduce an adaptive, reciprocity-based Machine Learning mechanism to estimate the answers of a variety of aggregate queries (AQs) avoiding the big data back-end. The mechanism learns from past analytical-query patterns while they develop solutions to correspond in the changes in queries' analytics and analysts' interests.

CHAPTER 3: DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

Nowadays, a tremendous number of IoT devices, computers, and applications produce data at a humongous rate and in large quantities. The produced data are stored into datacenters across several sites which are geo-distributed all over the world. In order to make possible the maintenance and utilization of such large collections of data the use of a database management system (DBMS) is required. Every site has its own right and local users, a separate local DBMS which may differ of the other sites and its own local data managers. The sites communicate and are connected using high-speed networks creating distributed systems. The union of the separated databases of the sites forms a distributed database. Therefore, when we use the term distributed database management system (DDBMS), we are referring to the software system that allows a user to manage the distributed database. An important characteristic of a DDBMS is that gives the user the view of a united database while the data is physically distributed in the datacenters of the computer systems which participate in the distributed system. In other words, a DDBMS is logically integrated but physically distributed. Figure 1 presents a basic structure of a distributed database architecture. In the next two paragraphs, we describe (i) the types of distributed databases; (ii) the meaning of parameters in which the classification of DBMS architectures developed and (iii) three categories of architectural models for DDBMS i.e., the Peer-to-Peer DBMS, Client-Server Systems, Multidatabase Systems (MDBSs) [8], [9], [10].

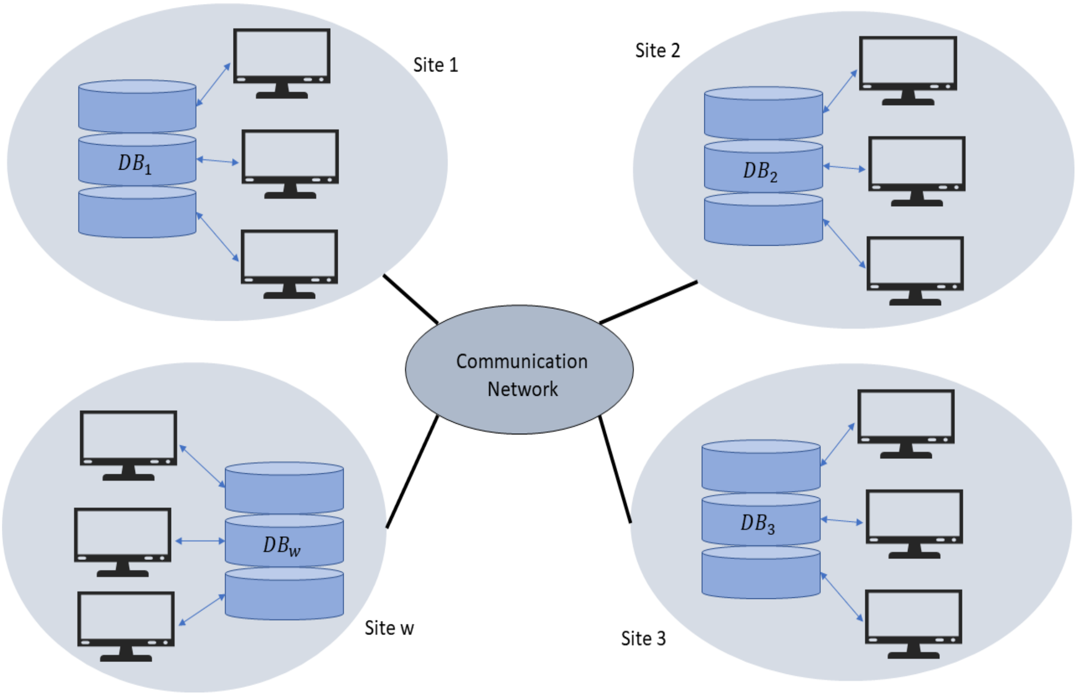


Figure 1: Distributed database architecture

3.1 TYPES OF DISTRIBUTED DATABASES

In this paragraph, we present the types of distributed databases and their properties. There are two categories of distributed databases based on the kind of DBMS software they use. The first and simplest category is the Homogeneous DDBMS. In this category, all the sites of the distributed system running their applications on the same DBMS software. Additionally, every site knows the existence of all the other sites, and they work together to process user's request. Homogenous databases permit users to have access data from each of the databases like it was a single database. All the aforementioned characteristics of homogenous databases making them easy to manage. On the other hand, in heterogeneous DDBMS, the sites are controlled by a variety of different DBMSs. The sites work autonomously and are connected with such way to allow users the access to data from the other sites. Also, each site may not be informed of the existence of the other sites, which results to the limited co-operation in the users' requests processing. Summarizing, in heterogeneous DDBS each site operates like an independent and autonomous centralized DBMS which has its own local characteristics i.e., users, transactions and database managers.

3.2 PARAMETERS FOR THE DEVELOPMENT OF DISTRIBUTED DATABASES MANAGEMENT SYSTEMS ARCHITECTURES

Taking into consideration that a DDBMS consist of computer systems, there exist three parameters based on which the DBMS architectural models are created. These parameters determine the distribution, the autonomy and the heterogeneity of the computer systems. In [9], the authors represent these parameters in as dimensions of a 3-dimensional space as it is presented in Figure 1. We can easily distinguish the Parallel DBMS, NoSQL, NewSQL DBMS, Peer-to-Peer DBMS, Client-Server Systems and MDBSs.

3.2.1 AUTONOMY

One of the first parameters we have to define before the implementation of a DBMS is the autonomy. More specifically, when we refer the term autonomy for a DBMS we refer to the distribution of control in the system i.e., in the degree of independence in the operation of a DBMS. We can classify the DBMS systems in three categories based on the degree of autonomy, systems which characterized by tight integration, semi-automatous systems and systems with total isolation.

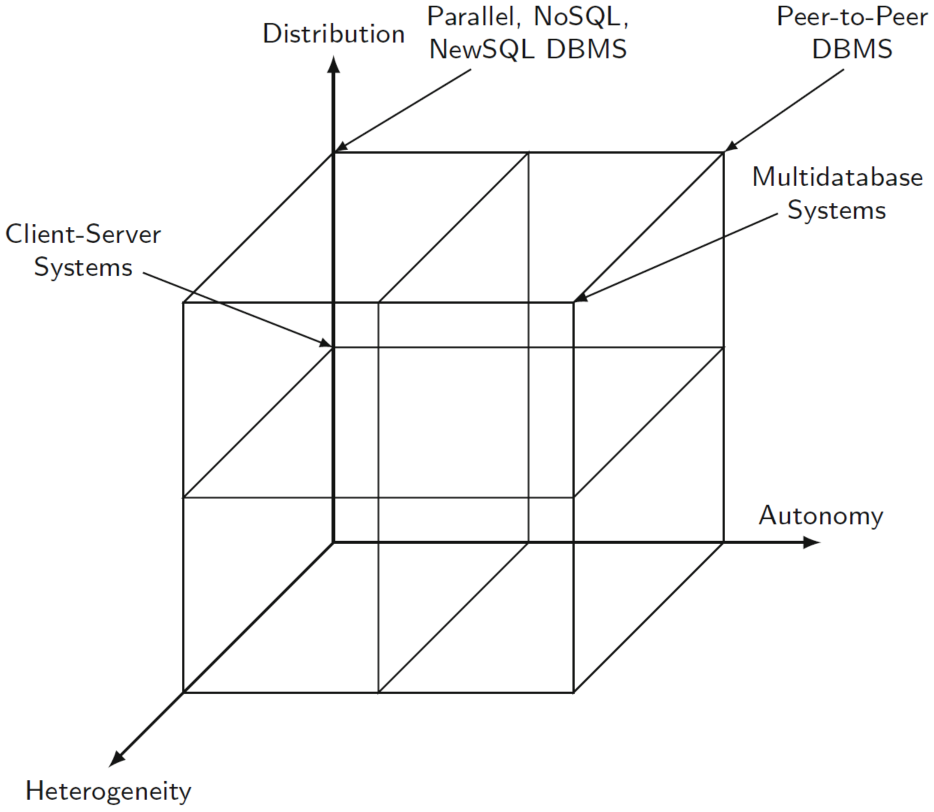


Figure 2: Parameters for the development of DBMS architectures *[9]*

The first class of the above taxonomy is the tight integrated systems which has the lowest degree of autonomy of all classes. In this type of DBMS, a single-image of the entire database is available to all users who want to access data and share them even the data belong to multiple databases. The users have the view for the data that are logically centralized in one database. An important component of DBMS is the data manager which is a set of computer programs that provide the database management functionalities. In DBMS with tight integration, the data managers are applied with such way to make one of them responsible for control of the processing of users' requests even the requests can be serviced by more than one data managers. Additionally, the data managers of tightly integrated systems despite their ability to operate as independent DBMS do not operate as such.

The second category is the semiautonomous systems. The semiautonomous systems are comprised by DBMS which usually has independent operation. However, these systems take part in a cooperative group to make their data from the local database sharable with the other systems in the same group. In order to achieve that they need to modify their operations in semiautonomous operation and specify which parts of their own database will make sharable to other users of different DBMS.

The third and last category, based on the autonomy as it was defined before, is the total isolation. This category includes individual DBMSs which do not know either the existence of other DBMSs or the way to communicate with them. Consequently, the processing of user transactions, which requires data from multiple databases, is extremely difficult due to the absence of global control over the execution of individual DBMSs.

3.2.2 DISTRIBUTION

One additional parameter in the development of DDBMSs is the distribution. This parameter states the distribution of the data across the computer systems in a DDBMS. However, this parameter does not change the view of the users for the data as one undivided set. Taking into consideration only this parameter, we can identify three different architectures i.e., client/server distribution, peer-to-peer distribution and the option of non-distributed architecture.

In the first category, the servers are in charge with the responsibility of execution of data management processes while the clients make the application environment and user interface is available in users to perform data management tasks. Both clients and servers are responsible for the communication processes. Due to its characteristics the client/server architecture constitute an example of distributed functionality. In contrast with the former architecture in peer-to-peer architecture each machine has full DBMS operation, and it can cooperate with other machines to execute queries, transactions and other data management processes. For this reason, the peer-to-peer architecture also called fully distributed.

3.2.3 HETEROGENEITY

The third parameter in the development of DBMSs architectures is the heterogeneity among the distributed systems. This heterogeneity results from various reasons, such as dissimilarities in the hardware of DBMSs, networking protocols etc. One of the most important reasons that causes heterogeneity, is the way that data are represented by various modeling tools in different databases, on account of the characteristics of individual data models. Also, the usage of data models is strongly related to the choice of the query languages, which may provoke heterogeneity. The heterogeneity is created by the usage of different query languages for the access in data, not only in different but also in the same data models.

3.3 ARCHITECTURES FOR DISTRIBUTED DATABASE SYSTEMS

3.3.1 CLIENT/SERVER SYSTEMS

The first of the architectures that we describe is the client-server architecture. This architecture is developed upon the separation of functionality for the server operation and for the client respectively. Based on the previous idea, a two-level architecture has been created, that facilitates the management of the distribution and consequently the complexity of modern DBMSs. The client-server architecture is divided into three categories: single-tier client-server, two-tier client server and three-tier client server.

The single tier client-server architecture in DBMSs is the simplest architecture that can be adopted. In this architecture, the client, the server and the database are located in the same machine. Users use terminals to send SQL queries into the central computer where the database is stored to retrieve the needed data. However, this type of architecture is rarely used in practice due to the humongous burden of the computer system which host the database and the terminals.

The second category has two different implementations: (i) multiple clients/ single server, (ii) multiple clients/ multiple servers. The former case is the simplest implementation of the two-tier client/server architecture because there is only one server that is used for the answering of the queries. Additionally, the database and the appropriate software for the management of the database are stored in the server as it is also applied in centralized systems. Nonetheless, the multiple client/server architecture differs from the centralized databases in many characteristics as the execution of the transactions and the management of caches. The latter case of client/server architecture is more complex due to the participation of multiple servers in the system. In this architecture, there are two management strategies. In the first management strategy, every client manages its connection to the appropriate server. While in the second strategy each client knows of only the server with which is connected, which then co-operates with other servers when this is necessary. The main difference between the aforementioned strategies is detected in the loading of the extra responsibilities in clients in the first strategy, and in the servers in the second strategy. Figures 3 & 4 present the multiple clients/ single server implementation and multiple clients/ multiple servers respectively.

Figure 5 presents one additional architecture which is the three-tier client-server architecture. This architecture is an extension of two-tier client-server architecture in order to make the architecture of client-server more efficient and flexible. In this type, an extra tier which is called the application server is added between the client and the database server. Consequently, the client communicates with the server through the application server to access the database and the query processing and transaction management take place.

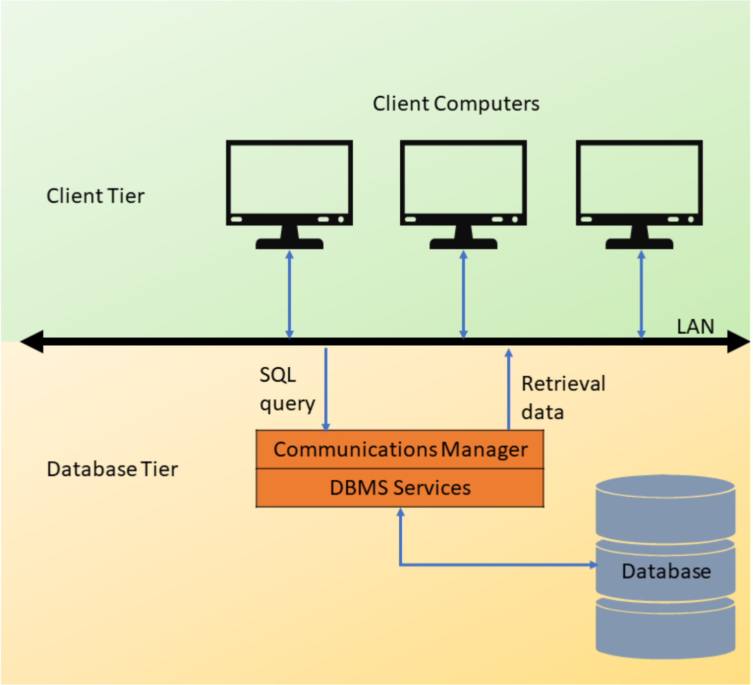


Figure 3: Multiple Clients/ Single Server implementation

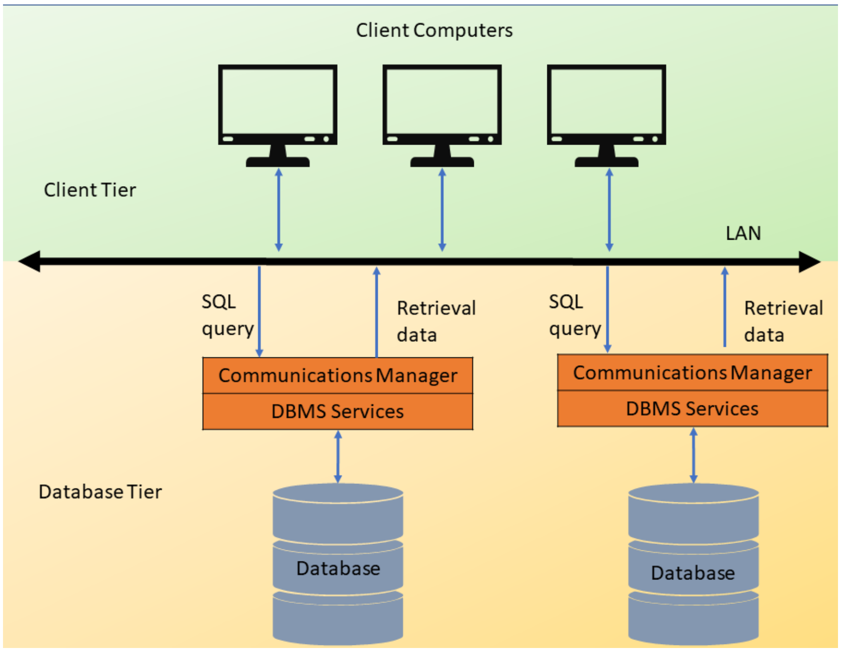


Figure 4: Multiple Clients/ Multiple Servers implementation

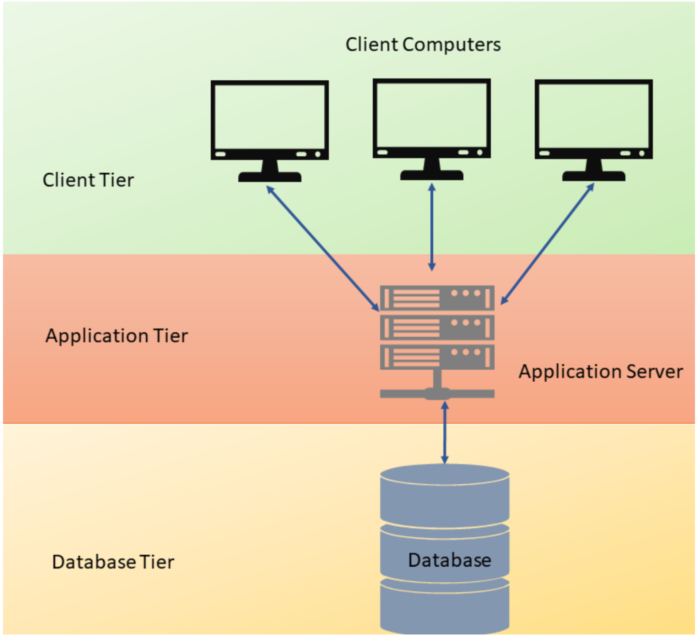


Figure 5: Three- tier client-server architecture

The last client-server architecture is the n-tier client-server. This approach is extended by the involvement of multiple database and application servers. The application servers serve one or more applications and the same stands for the database servers. Additionally, the interface to the application occurs using a load balancer that conveys the queries of the users to the appropriate servers.

3.3.2 PEER-TO-PEER SYSTEMS

In this architecture, every peer acts as a client and a server to serve the requests of users through cooperation and sharing data with other peers. In peer-to-peer systems, the centralized database has its own schema definition which is called global conceptual schema (GCS) and depicts the global logical view of data. Each site of the DDBMS has stored a part of the centralized database in local databases which has a different schema from the centralized database. The schema which expresses the logical data organization for every local database in each site is called the local conceptual scheme (LCS). We have to note that the physical data organization on each site may be different and as a consequence, each site has a different Local Internal Scheme (LIS). Additionally, the schema definition that depicts the user view of data is called External Schema (ES). Users sent queries that are created based on the GCS. The DDBMS transforms the global queries into a set of local queries based on the query language of each local dataset. Afterwards, the DDBMS components execute the set of local queries at different sites which nevertheless communicates among them. There are two major components which is used for the handling of the incoming queries; the user processor and the data processor. The first component i.e., user processor is responsible for the interaction with the users and the second component has the responsibility to deal with the data storage. The user processor consists of the following four sub-components:

- **User interface handler**: it has to interpret the incoming queries and other commands from the user and transform the results into appropriate format to answer the queries.
- **Semantic data controller**: this sub-component checks the integrity constraints, as defined by the database elements, by using the GCS. Also, it inspects the authorizations in order to allow access to the appropriate database.
- **Global query optimizer and decomposer**: It tries to find the best possible execution strategy for the user queries or requests. Its goal is to minimize as much as possible the cost function and transform the GCS queries to LCS queries.
- **Distributed Execution monitor**: it organizes the distributed execution of the user queries. Basically, it is a transaction manager which during the execution of the incoming query communicate with the transaction managers of other sites.

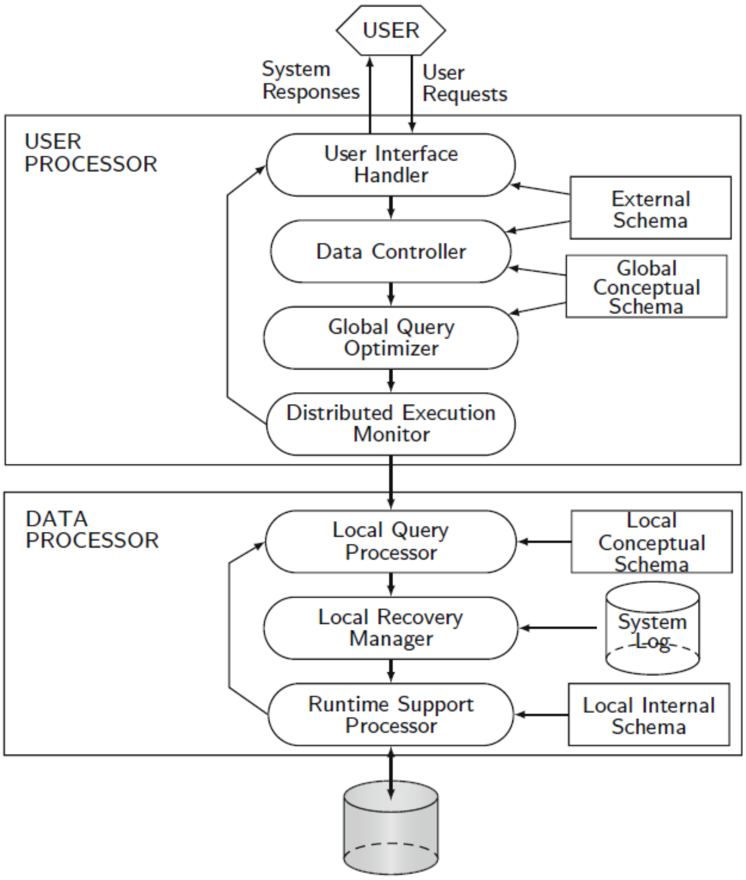


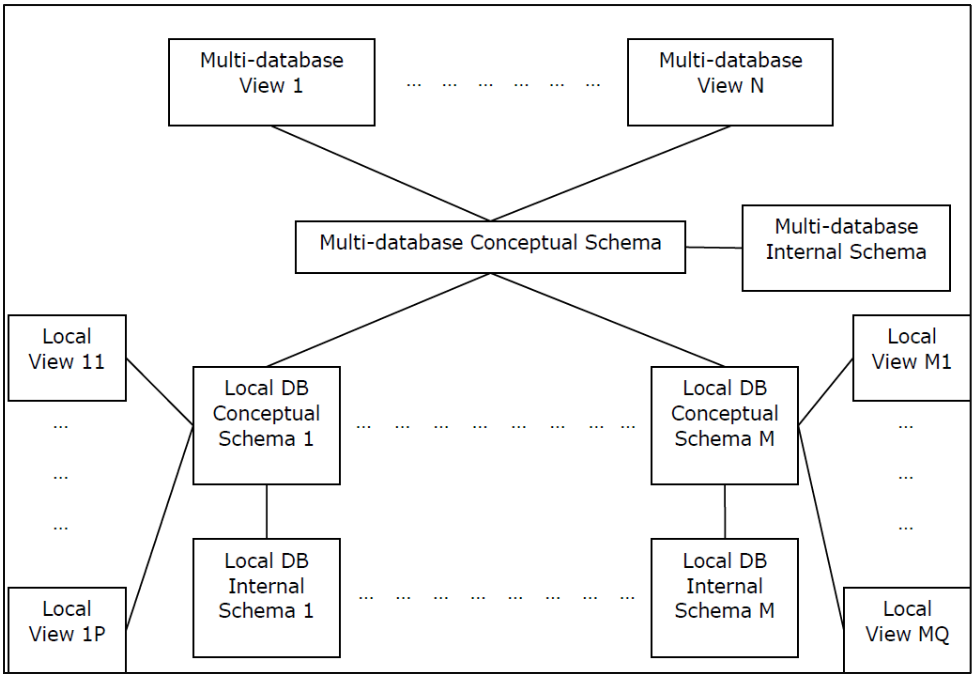
Figure 6: Peer-to-Peer DDBMS architecture

The second of the components in a Peer-to-Peer DDBMS is the data processor and is comprised by the following elements:

- **Local query optimizer**: It has similar functionality with the access path selector. The goal of this sub-component is to choose the best access path to retrieve or take access to any data.
- **Local recovery manager**: This sub-component is responsible for the maintenance of the consistency of the local databases even when failures occur.
- **Run-time support processor**: It accesses the distributed database physically based on the strategy that the local query optimizer proposed. Additionally, the run-time support processor is the responsible interface for the communication with the operation system and contains the database buffer manager for the maintenance of the main memory buffer and for the management of the data accesses.

3.3.3 MULTIDATABASE SYSTEMS

The last of the presented architecture for DDBMS is the Multidatabase systems (MDBSs). This type of architecture consists of two or more autonomous database systems and can be described using six levels of schemas. The first schema definition which is called Multi-database View Level describes the view of different users which consist of the subsets of the initial integrated distributed database. The second level in the MDBSs is the Multi-database Conceptual Level which represents integrated multi-database which consists of global logical multi-database structure definitions. Another level in the six-level representation is the Multi-database Internal Level. This level gives a view of the data distribution in different sites and multi-database to local data mapping. The first level which refers into the local databases is the Local database View Level, it expresses the public view of local data. An additional schema is the Local database Conceptual Level that shows the way that the data are organized in the local database at every site. The last level of the six level of schemas is the Local database Internal Level that expresses the organization of the data in physical level at every site. In MDBSs there are two different design approaches. The different between these two approaches is the existence of the multi-database conceptual level. Figures 7 & 8 present these design approaches, respectively.



Multi-database View 1
… … … … … …
Multi-database View N
Multi-database Conceptual Schema
Multi-database Internal Schema
Local View 11
…
…
…
Local View 1P
Local DB Conceptual Schema 1
… … … … … … … …
Local DB Conceptual Schema M
Local DB Internal Schema 1
… … … … … … … …
Local DB Internal Schema M
Local View M1
…
…
…
Local View MQ

Figure 7: MDBS with Multi-database Conceptual Level

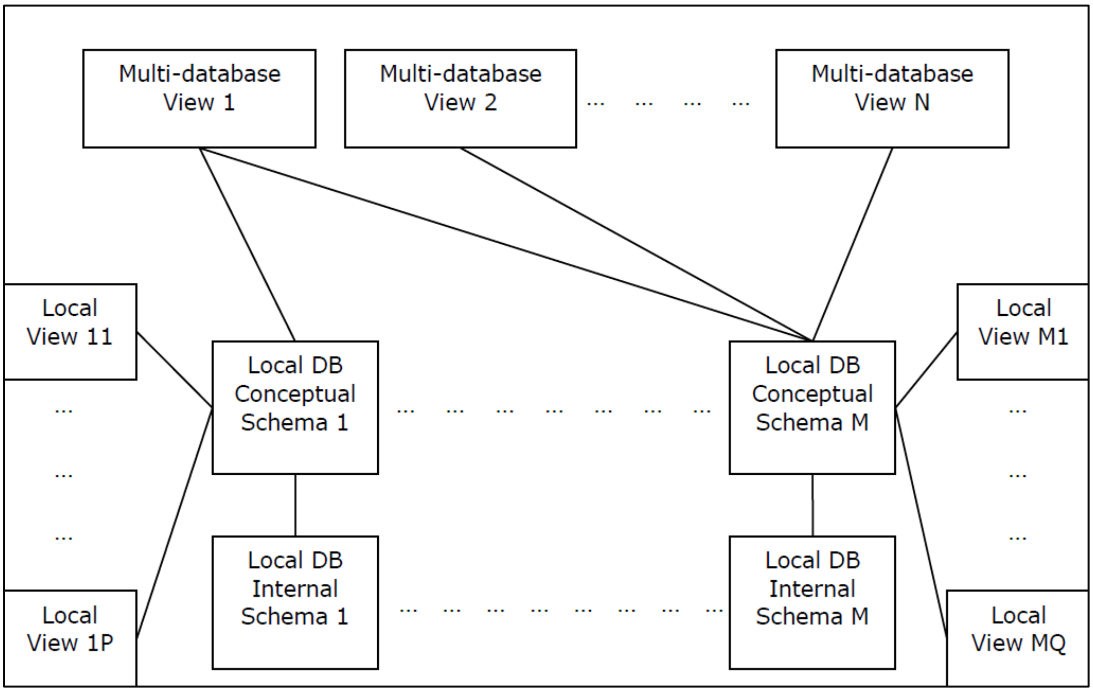


Figure 8: MDBS without Multi-database Conceptual Level

3.4 ADVANTAGES OF DISTRIBUTED DATABASES MANAGEMENT SYSTEMS

The usage of DDBMSs instead of classical centralized database systems has several advantages. In this paragraph, we present four of the most important advantages. The first of the presented benefits of the DDBSs is the potentiality of system extension. More specifically, the attempt for an extension in centralized database systems requires the interruption of the functionality in the database and a great amount of effort. Nevertheless, the equivalent effort in DDBMSs is much simpler since it requires the addition of new computer systems and data to the newly created local databases at the respective new sites. Afterwards, the newly created systems connect to the DDBMS without having to interrupt the operations that take part at that time. Additionally, the DDBMS functionality is much more durable in the failures than centralized systems that making the DDBMS more reliable. The architecture design of the DDBMS allows them to continue their functionality despite the occurrence of failure at one or more components. Furthermore, one significantly important advantage is the better response in the user queries due to the distribution manner of data at the local databases of sites. Contrariwise, in the centralized databases systems, every query has to pass through the central computer increasing significantly the responding time. The last but not least advantage of DDBMSs against centralized databases systems is the lower cost of communication. In case that an incoming query requires data that is stored in local database where it is mostly used then the communication cost for the data retrieval is minimized.

CHAPTER 4: DATA CLUSTERING ALGORITHMS

Clustering is a significant process especially when we have to retrieve data which has to satisfy specific requirements. The clustering algorithms are divided into categories based on the way that it is used and the purpose which serve. This chapter includes the most widely used categories of clustering algorithms and refers some of the most well-known algorithms of these categories [11], [12].

4.1 PARTITION CLUSTERING ALGORITHMS

Partition clustering algorithms constitute one of the most prominent categories of clustering algorithms. These algorithms receive as input a set of h data points and perform cluster analysis by partitioning these data into v clusters, which is usually pre-defined and must be less or equal to h i.e., the size of the dataset. The created clusters are not overlapped, and every cluster has at least one data point as a member. More specifically, the partitioning-based clustering algorithms perform an initial clustering and afterward is based on an iterative approach which is trying to minimize as much as possible the objective function. In the initial partitioning process, v data points are chosen to become the points on which the algorithm will base to create the clusters. The choice of these initial points is a significant consideration for the quality of clustering and the performance of the algorithm. While the distance between the initial points should be as bigger as possible in order to be succeeded the better shaping of clusters. This subject has constituted a subject for many research activities which try to find the optimal solution. [13]

The most representative algorithm of this category is K-Means which will be described in paragraph 5.1 together with the description of the Fuzzy C-Means. The Partitioning Around Medoids (PAM) algorithm is usually used from the research community. PAM is based on k-medoid method for clustering and is more durable in noise and outliers than K-Means. However, PAM algorithm does not work well for large scale datasets. The algorithm consists of two phases, the Build phase and Swap phase. In former phase the algorithm selects k medoids by choosing the k points which has the minimum cost, with cost being the sum over all distances to all other points. The selected k points are initially the representative objects for each cluster. In Swap phase the algorithm calculates the average dissimilarity to all non-selected objects to the k medoids. Afterward, the algorithm groups the non-selected objects to the closest medoid. The Swap phase is repeated until there are not exist better medoids [14].

One additional algorithm in this category is the CLARA algorithm which created to overcome the disadvantages of PAM and is based on sampling. The idea behind the CLARA is that if the samples are sufficiently random, the medoids of the sample approximate the medoids of the dataset. In the first step the algorithm divides the dataset into multiple subsets with a specific size. Following the algorithm applies the PAM algorithm on every subset and select the corresponding k medoids and assigns each observation of the whole dataset to the nearest medoid. Afterward, it calculates the dissimilarities of the observations to their nearest medoid. The algorithm retains the sub-dataset which has the minimum dissimilarity and performs further analysis in the final partition [14].

4.2 HIERARCHICAL CLUSTERING ALGORITHMS

Another category of clustering algorithms is the hierarchical clustering algorithms. This type of algorithms builds a hierarchical decomposition of the dataset which has to be clustered. There are two subcategories of hierarchical algorithms, the agglomerative and the divisive hierarchical algorithms. The categorization of the hierarchical algorithms in one of the aforementioned categories depends on the way that the hierarchical decomposition is created. The former category is also called the bottom-up category because the algorithms which belong in this category consider every data point as a separate cluster. Afterwards, the algorithms merge the clusters which are close using several metrics such as similarity, correlation, distance. The merging procedure is executed repetitively until all clusters are merged into one cluster which contains all the data points, or a termination criterion is reached. The second subcategory is called divisive and is a top-down approach. More specifically, the algorithms of this type begin their clustering process by setting all data points in one cluster. Then, in every iteration, the initial cluster is split into smaller clusters until a stopping criterion is reached. Hierarchical algorithms compared with partition clustering algorithms create clusters that are more informative. However, the hierarchical algorithms can't undo an applied step.

A typical hierarchical algorithm is the Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) [15]. This algorithm is based on distance and has the ability to cluster the data at the same time it scans them. Additionally, BIRCH during the clustering process take into consideration many factors such as the efficiency of time and space, the sensitivity of data input order, the accuracy etc. The clustering process of BIRCH use the notions of the Clustering Feature (CF) and the Clustering Feature Tree (CF-tree). The CF is a triplet (N, LS, SS) where N depicts the number of data which belong to the cluster, LS is the linear sum of data in the cluster and SS is the sum of squares of data in the cluster. The LS and SS are calculated by the equations 1 & 2 respectively. The CF-tree is a highly balanced tree in which are stored the clustering features of the hierarchical clustering. In a CF-tree there are two kinds of nodes the leaf nodes and non-leaf nodes where each non-leaf node contains the sums of the CFs of their children. The construction of CF-tree affected by three parameters, the branching factor B which define the maximum number of child nodes that can be connected to a non-leaf node, the maximum number of entries L in every leaf node and T which define the maximum diameter of subclusters stored at the leaf node of the CF-tree. The BIRCH algorithm consist of two phases as follows

- **Phase 1:** BIRCH scans the data and creates a CF-tree with respect to the parameters B, L, T.
- **Phase 2:** BIRCH adopts a clustering algorithm to cluster the leaf nodes. This procedure has as result the removal of sparce clusters as outliers and merge the dense clusters into bigger one.

$$LS = \sum_{i=1}^{N} r_i \qquad (1)$$

$$SS = \sum_{i=1}^{N} r_i^2 \qquad (2)$$

Chameleon is also a widely used hierarchical clustering algorithm. Chameleon adopts a k −nearest neighbor graph approach to build a sparce graph. In sparce graph each data point represented as node of the graph and is connected with the top-k similar neighbors. The edges of the sparce graph are weighted to show the similarity between the data points. Afterwards, Chameleon applies a graph partitioning algorithm to split the k-nearest neighbors graph into a large number of subclusters such that minimizes the edge cut. Particularly, the algorithm tries to partition a cluster Ω with such way to minimize the weight of the edges that would be cut should Ω be split into Ω_i and Ω_j, for this reason the Chameleon relies on the absolute interconnectivity between clusters Ω_i and Ω_j. In the last phase of the Chameleon an agglomerative hierarchical clustering algorithm is adopted to merge the subclusters taking into consideration the similarity among the subclusters. The similarity of the subclusters is measured using the relative interconnectivity and the relative closeness of the subclusters [16].

4.3 DENSITY BASED CLUSTERING ALGORITHMS

The category of density-based clustering algorithms introduced to discover cluster with arbitrary shape. The clustering process of algorithms of this category is based on the density of data points in an area. Especially, these algorithms assign data points in a cluster as long as the number of points i.e., density, within a radius is higher than a threshold. A significant characteristic of the density-based algorithms is the ability to detect noise and outliers in the examined dataset.

One of the most famous algorithms in this category is the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [17], [18]. The clustering process of DBSCAN requires the definition of two parameters i.e., the minimum number of data points inside the neighborhood of a point (minPts) and the radius (ε) which defines the area around a data point that constitute the neighborhood of a data point. The DBSCAN based on the two aforementioned parameters categorizes the data points as core point, border point and noise point. Core point is the point that the number of data points in its neighborhood is greater or equal to minPts. A border point is the point where the number of the points in its neighborhood is less than minPts but at least one of them is core point. The points which are neither core points nor border points are characterized as noise points. The steps of the DBSCAN algorithm are described below:

- Step 1: The algorithm labels all points as core or noise points.
- Step 2: It relabels the noise points that have as neighbor at least one core point as border points.
- Step 3: It marks all core and border points as unvisited.
- Step 4: It selects an unvisited core point and creates a cluster.
- Step 5: The algorithm starts from the selected core point and groups to the same cluster the core points that are within the radius ε of each other and the points that belong in the neighborhood of the core points of the cluster. Each point that is added to a cluster is marked as visited.
- Step 6: It returns to the step 4 until all points are part of a cluster.

Another algorithm in this category is the Ordering Points To Identify the Clustering Structure (OPTICS) [19], [20]. This algorithm generates a distance profile that displays the dataset's density structure and may be used to extract clusters using at least two parameters: a distance matrix and a number of neighbors. The algorithm iteratively explores point neighborhoods in order of lowest to highest core distance, i.e. the maximum distance between a point and a given number of its nearest neighbors, and returns the orders and reachability distances of successive points, i.e. the maximum distance between the point's core distance and the distance from it to the previous point. Valleys with low reachability distances symbolize clusters and are separated by peaks, or places with high reachability distances.

CHAPTER 5: PRELIMINARIES

In this chapter we present the algorithms that we involve in this research. More specifically, we describe the K-Means (KM) algorithm, the Fuzzy C-Means (FCM) etc.

5.1 K-Means

KM is one of the most popular and widely used unsupervised clustering algorithms. This algorithm groups the given data into K clusters trying to minimize the following objective function:

$$J = \sum_{i=1}^{K} \sum_{p \in G_i} \|p - g_i\|^2 \tag{3}$$

The minimization of equation (1) is equivalent to the minimization of distance of points in a cluster G_i with the centroid g_i. We consider a set $D = \{\vec{p}_1, \vec{p}_2, \dots, \vec{p}_n\}$ which consists of n vectors each one can be considered as a d −dimensional point. KM has as goal to split the n vectors into K clusters, where $K \leq n$. The algorithm requires the number K of clusters is defined in advance. The clusters have to have been created in such way such that $G = G_1 \cup G_i \cup, \dots, \cup G_K$ where $G_i \subset G, \forall i \in [1, K]$ and $G_i \cap G_j = \emptyset,\ \forall\, i, j\ \in [1, K]$. Each cluster is represented by its centroid. The steps of the KM algorithm are described below [21, 22, 23]:

- At first, K points from D are randomly selected to be the centroids of the clusters.
- In the second step, the algorithm computes for every point in D the distance from every centroid. Hence, the algorithm assigned the examined point to the cluster with nearest centroid. The Euclidean distance is the most widely adopted metric for this step.
- Afterwards the algorithm recalculates the centroids of each cluster. The new value for each one of d dimensions is equal to the average value that the members have in that dimension.
- The algorithm returns to the second step in case the criterion function does not become the minimum, i.e., the clusters does not remain consistent.

We have to note that the centroids of the clusters may not contained in the dataset D, since it will arise from the previous iteration process. The time complexity of KM depending on three parameters; the number n of data vectors in D, the number of clusters K and the number of iterations l that the algorithm needed to cluster the data. Consequently, the time complexity is equal to $O(n \cdot K \cdot l)$ [23].

5.2 Fuzzy C-Means

The FCM algorithm belongs to the fuzzy clustering algorithms. Contrary to the hard-clustering algorithms, where each data point belongs only to one cluster, in fuzzy clustering the data points can potentially belong to multiple clusters. In the FCM, each data point is associated with a cluster by a membership value $u_{ik} \in [0,1]$. The membership value shows the similarity between the examined data point and the center of the respective cluster. The higher the membership value is, the higher the similarity is. Suppose we have a dataset D of n vectors each one has d dimensions the FCM outputs a $n \times \Gamma$ matrix **U** that shows with which cluster has the higher similarity and a set $C = \{c_1, c_2, c_3, \dots, c_\Gamma\}$ which contains the centers of the clusters. We have to note that the sum of each row in **U** must be equal to one. The FCM aims to minimize the following objective function:

$$J_m = \sum_{i=1}^{\Gamma} \sum_{k=1}^{n} (u_{ik})^m \cdot \|p_k - c_i\|^2 \qquad (4)$$

where m is the fuzzier parameter which controls the fuzziness of the clustering. The value of m is any natural number bigger than one i.e., $m \in [1, \infty)$. The algorithm requires to pre-define the fuzzier m, the number of clusters Γ and the stopping criterion value $\beta \in [0,1]$. An extensive research in the parameters of FCM is presented in [24]. The steps of the algorithm are referred below [21, 22, 25, 26]:

- Step 1: Choosing of the parameters m, Γ, β
- Step 2: Initializing the membership matrix **U**
- Step 3: Calculating the centers of every cluster in C
- Step 4: Updating the membership matrix **U**
- Step 5: Repeating steps 3,4 until the divergence is less than β
- Step 6: Output **U**, C

The updating of the membership value u_{ik} and the center for each cluster in FCM are calculated by the equations $(3), (4)$ respectively.

$$u_{ik} = \frac{1}{\sum_{j=1}^{\Gamma} \left\{ \frac{\|p_k - c_i\|}{\|p_k - c_j\|} \right\}^{\frac{2}{m-1}}} \qquad (5)$$

$$c_i = \frac{\sum_{k=1}^{n}(u_{ik})^m \cdot p_n}{\sum_{k=1}^{n}(u_{ik})^m} \quad (6)$$

The time complexity of the FCM is affected by the number of data n that need to be clustered, the number of clusters Γ which have to be created and the number of iterations l that the algorithm demands to complete the clustering process.

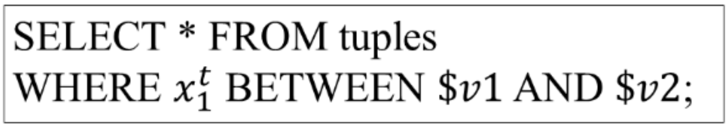
CHAPTER 6: PROBLEM DESCRIPTION

In our scenario, we consider the set $DB = \{DB_1, DB_2, \dots, DB_w\}$ of geo-distributed DBMSs and a server (SV). Also, we suppose that a group Z_i of Internet of Things (IoT) devices is connected with a DBMS DB_i such that each Z_i to be connected with only one DB_i and vice versa. The IoT devices collect and report data into the respective DB_i with form of multivariate vectors i.e., $X_j^t = [x_1^t, x_2^t, \dots, x_d^t]$, where the index j expresses the IoT device that reported the vector and the index t shows the time instance that the vector was reported. The DBMSs receive the multivariate vectors and store them in appropriate format to be ready for further processing activities. Table I shows the format in which the data is stored in every DBMS in our scenario.

Table 1:The appropriate format in which the DBMSs store the data

Time instance	1^{st}IoT device	2^{nd} IoT device	…	N^{th} dimension
t	X_1^t	X_2^t	…	X_N^t
$t+1$	X_1^{t+1}	X_2^{t+1}	…	X_N^{t+1}
…	…	…	…	…
$t+W$	X_1^{t+W}	X_2^{t+W}	…	X_N^{t+W}

The SV communicates and has access in the DBMSs by using the network as the Figure 1 depicts. Many applications and tasks need a set of data which are stored distributed in different databases to perform their operations. The SV receives queries $Q = \{q_1, q_2, \dots, q_z\}$ from these applications and tasks. We consider that every vector can be represented as a point in a d-dimensional space. Every query q_i requires a number Φ of points as an answer. A q_i can contain range selection operators for one or more dimensions to create the boundaries of the area in which the data are located. For example, a simple query to the SV described as follows:



```
SELECT * FROM tuples
WHERE x_1^t BETWEEN $v1 AND $v2;
```

Figure 9: An Example of a query

The above example of query needs all the tuples where the value of the first dimension is range ($\$v1, \$v2$). The SV tries to detect the appropriate data as answer to the incoming queries of the tasks and applications in the minimum possible time. In this research we propose a mechanism for the detection of the data that the queries need. Our mechanism is ‘activated’ every time a query is sent by a user. More specifically, we focus on a hierarchical clustering process where we try to group the incoming query with previous received queries. The proposed model performs two types of clustering: (a) a fuzzy clustering where the incoming queries are assigned into one or more cluster based on a membership function; (b) a hard clustering to identify the subspace where the required data are located.

CHAPTER 7: ADOPTED METHODS IN OUR RESEARCH

In this chapter, we describe the methods that we adopt for the retrieval of appropriate data as answer the incoming queries. Each method is evaluated through the experimental evaluation process.

7.1 BASELINE METHOD

The first of the methods for the data retrieval is the baseline method (BM). This method is executed every time that a query is coming to the SV. The SV scans the data that exist in the distributed databases and choose those data that satisfy the query. More specifically, this method detects all the required points for each incoming query and is the optimal solution as far as the error is concerned. However, it needs a lot of time to scan for every query all the data that exist in the DDBs. This method is based on the searching on the DDBs for the appropriate data which satisfy the incoming query. Figure 10 gives a view about the execution of this method.

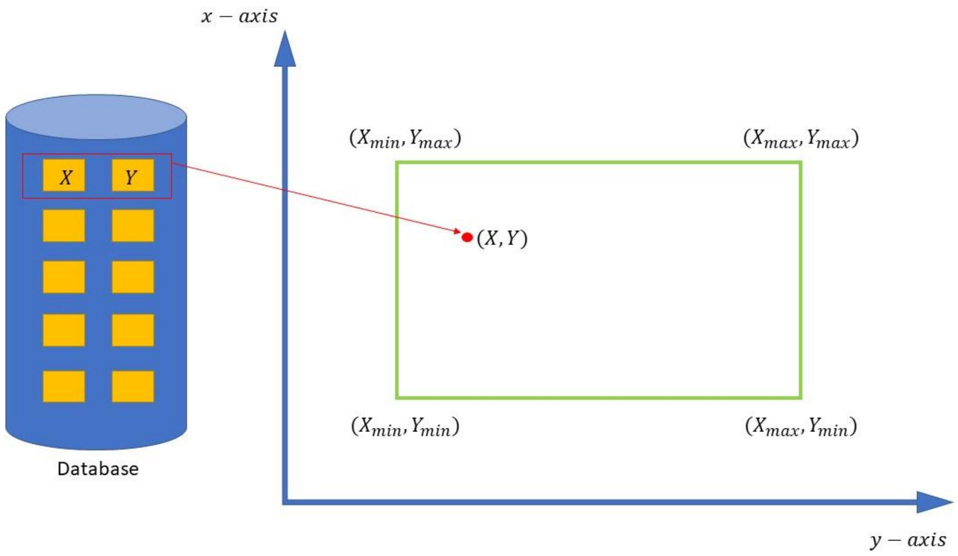


Figure 10: Baseline Method representation

7.2 HARD CLUSTERING BASED METHOD

The second method that we adopt in our research is called Hard Clustering Based Method (HCBM). This method uses a different approach for the detection of the appropriate data for every incoming query to the SV. Especially, this method is based on the clustering of the 'similar' queries, that the SV receives during a period of W time instances, in order to retrieve the appropriate data for the incoming queries. Initially, the HCBM receives a number z of incoming queries i.e., $Q = \{q_1, q_2, \dots, q_z\}$ and applies the BM method to find the data which are required for the execution of the incoming query. Afterwards, the HCBM applies the KM algorithm to cluster the queries which were sent in the SV in clusters to group the queries which has similar requirements. After the completion of the training phase, the HCBM method is ready to serve every incoming query in the server. More specifically, when a query is sent to the server, the HCBM is 'activated' and finds the top-k nearest/similar clusters to the incoming query adopting a distance/similarity measurement. In our experiments, we use the Euclidean distance metric. The HCBM is based on the detected top-k clusters and retrieve the data points which satisfy the queries which belong in these top-k clusters.

7.3 HIERARCHICAL MIXED CLUSTERING METHOD

Our proposed model is named Hierarchical Mixed Clustering Model (HMCM). This model focuses on the similarity more than the HCBM, relying on a hierarchical clustering. HMCM creates a set of clusters by the incoming queries using a soft clustering algorithm and then groups the queries which belong to the clusters into a set of subclusters by adopting a hard clustering algorithm. The proposed model consists of two phases, the 'warming' phase, and the 'performance' phase. The first phase of the HMCM model is a warm-up period in which are created the clusters and the subclusters. The clusters and subclusters which are arisen from the hierarchical clustering will be used to retrieve the appropriate data for the service of the incoming queries in SV. More specifically, when a query is sent from the user to the SV during the warm-up period, the HMCM model applies a sequential scanning in the entire database to find the appropriate data points for the execution of the query. After the detection of the required data points for every query, the HMCM performs the hierarchical clustering. Initially, it uses the FCM to create a set of clusters $C = \{C_1, C_2, C_3, \dots, C_\Gamma\}$ of the queries that were sent to the SV in previous W times. Afterward, HMCM divides each cluster in C further into a set of $G = \{G_1, G_2, \dots, G_K\}$ using the K-Means algorithm. In the second phase, the HMCM is activated every time a user sends a query to the SV. Firstly, the proposed model detects the top-k cluster whose members has same requirements with the incoming query using a similarity metric, in our case the Euclidean distance. After that, the HMCM detects the top-ℓ subspaces with which the incoming query has the higher similarity. A modification is used in order to achieve more accurate results in the retrieval of the appropriate data, i.e., we adopt the approach of overlapping between the representations of queries. The representation of each query sent by the user is the area of points that it needs as answer. For the purposes of the overlap calculation, we propose the following metric.

$$\text{AOM } (q_{inc}, q_{member}) = \frac{q_{inc} \cap q_{member}}{Incoming\ query\ area} \qquad (7)$$

In the Area Overlap Metric (AOM) the numerator is the overlapping area between two queries. The q_{inc} is the incoming query, while the q_{member} is the query which is member of one of the top-ℓ subspaces of one cluster which belongs to top-k clusters. The denominator is the area which contains the required data points for the incoming query. The result of the AOM indicates the percentage of the q_{inc} area covered by the area of the q_{member}. Hence, the HMCM, after the detection of the appropriate clusters and subspaces, examines the members of the detected subspaces to find the queries with which the AOM overcomes a threshold θ and retrieves only the data points that belong to them. This approach of selection of the members gives the ability to obsolete queries which belong into the same cluster, but they do not require data from a common area. Also, it offers the chance not to take into consideration queries with which have common area, but the overlapping is under the threshold and, this way, prevents the HMCM to increase the error in its predictions. Figures 11-26, we present the overlap cases between the areas of two queries.

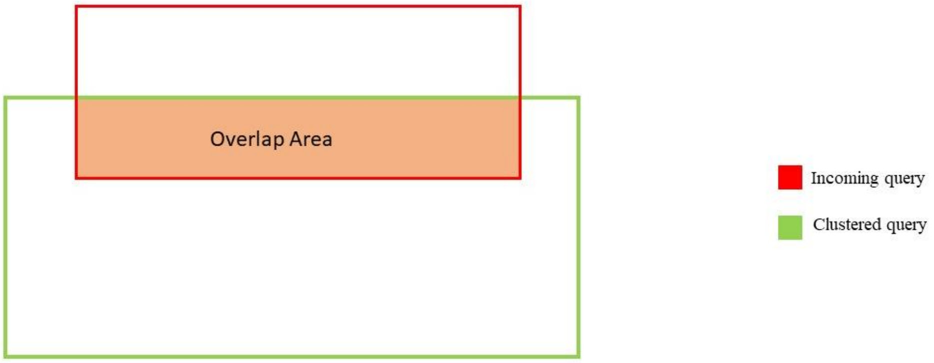


Figure 11: First overlap case

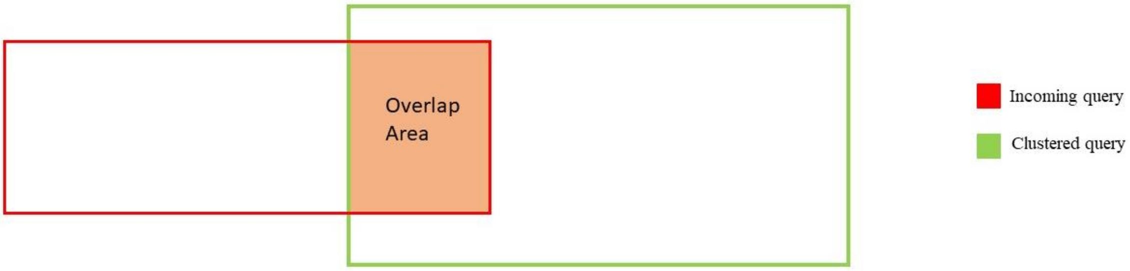


Figure 12: Second overlap case

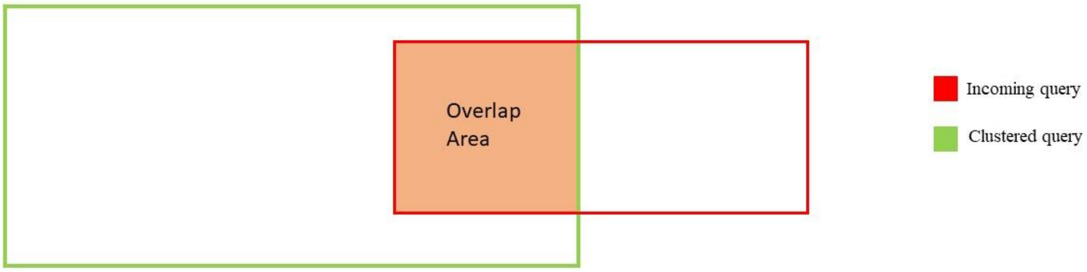


Figure 13: Third overlap case

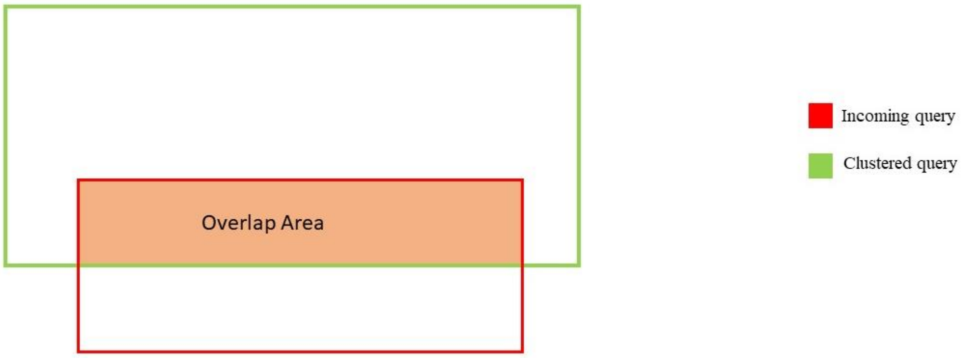


Figure 14: Fourth overlap case

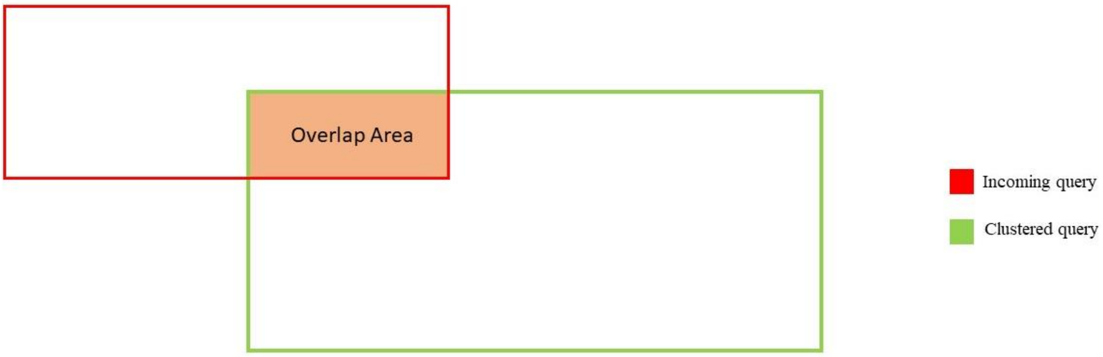


Figure 15: Fifth overlap case

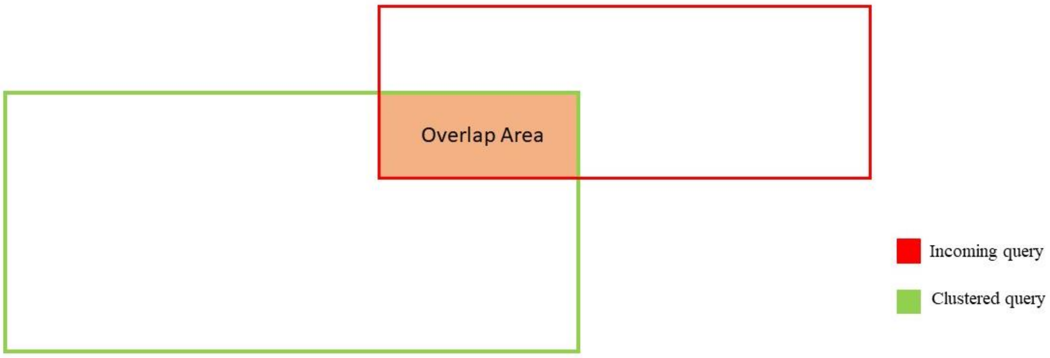


Figure 16: Sixth overlap case

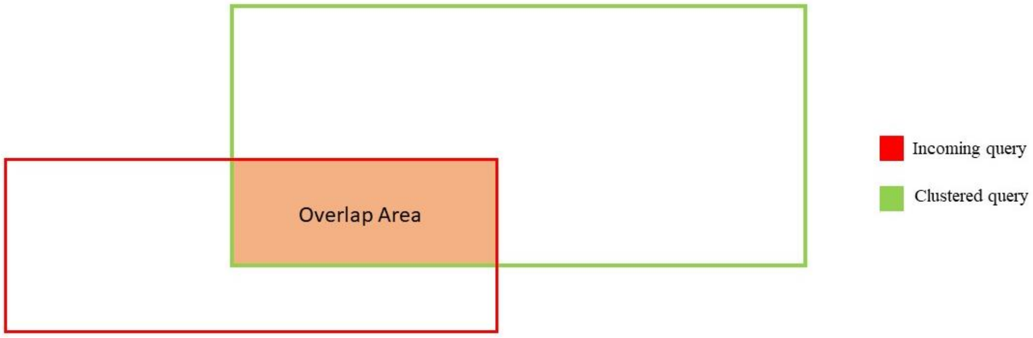


Figure 17: Seventh overlap case

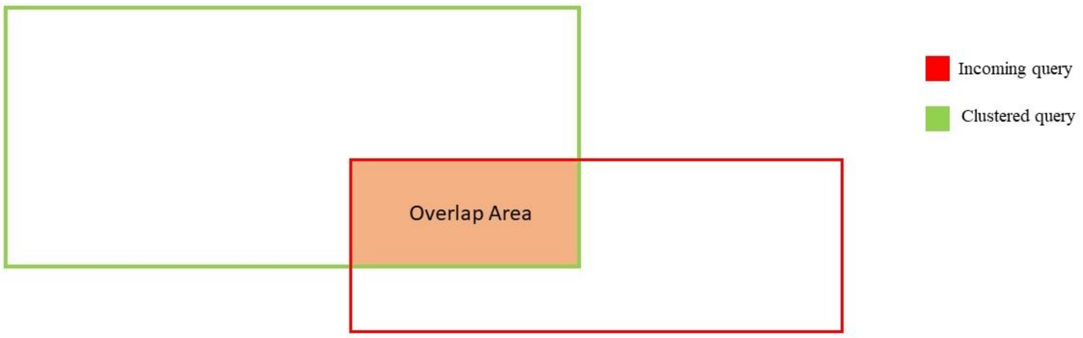


Figure 18: Eighth overlap case

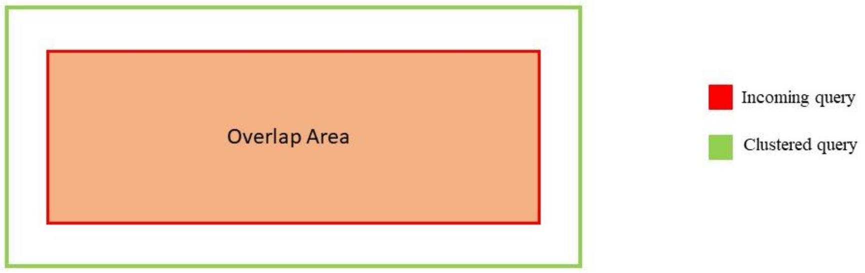


Figure 19: Ninth overlap case

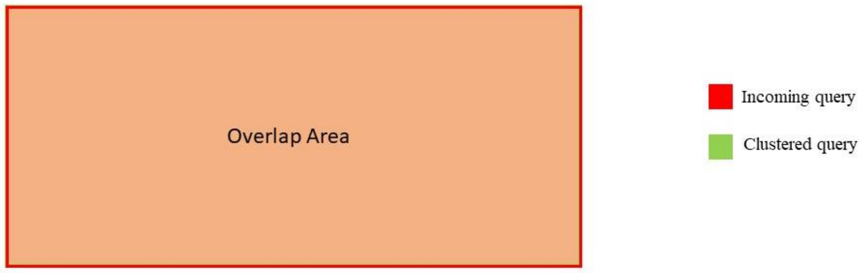


Figure 20: Tenth overlap case

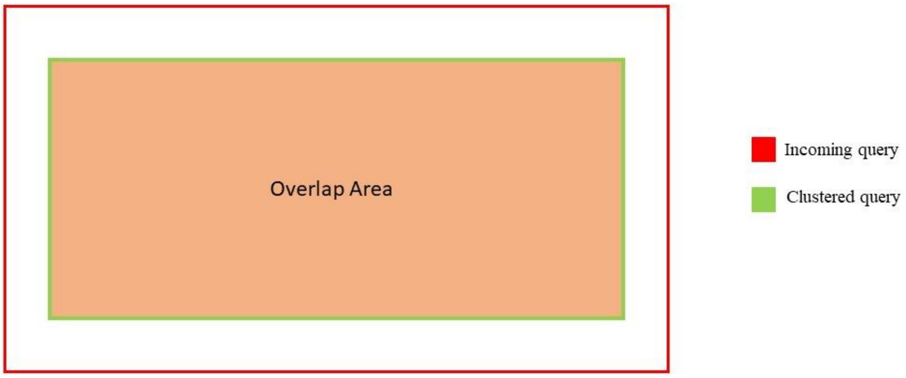


Figure 21: Eleventh overlap case

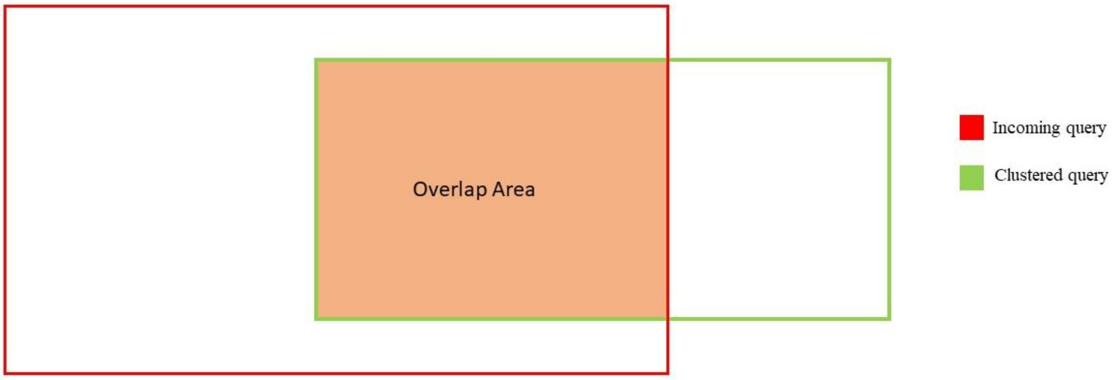


Figure 22: Twelfth overlap case

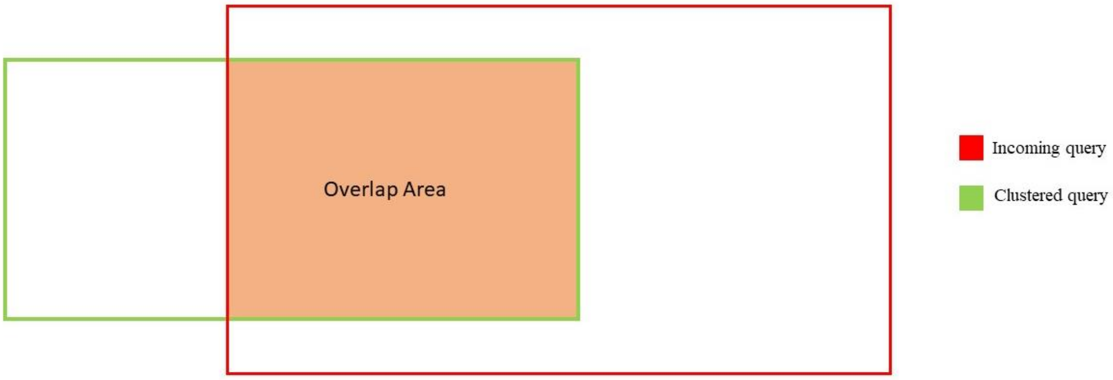


Figure 23: Thirteenth overlap case

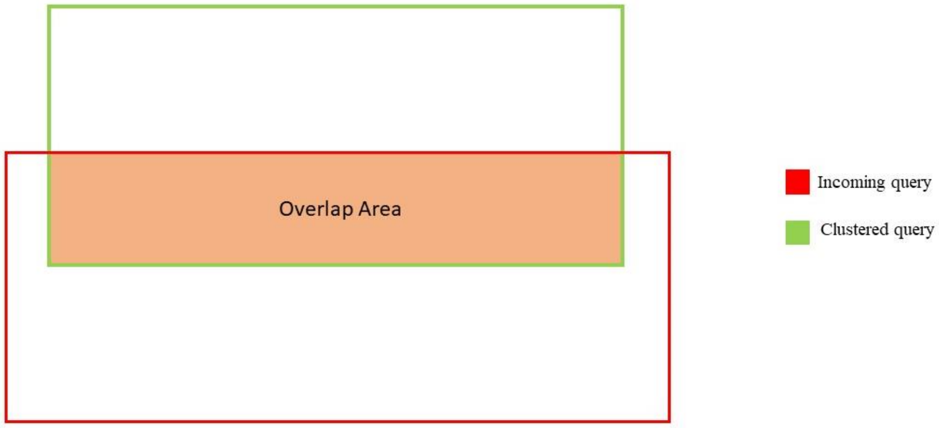


Figure 24: Fourteenth overlap case

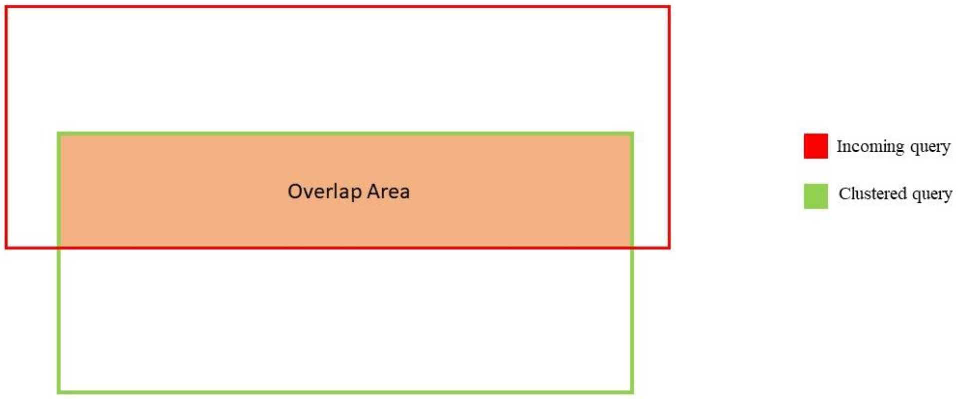


Figure 25: Fifteenth overlap case

CHAPTER 8: EXPERIMENTAL EVALUATION

The experimental evaluation of the proposed model is relying on the Query Analytics Workloads Dataset[1]. The dataset contains three files of range/radius query workloads from Gaussian distributions over a real dataset. In our experiments, we focus on range queries, and are based on the file Range Queries Aggregates to create three datasets which are named Warming Dataset (D_w), Dataset of two-dimensional points (D_{2d}) and Test dataset (D_T). Each range-query in the file is stored in the following format $\{X, Y, Xr, Yr, Count, SUM, AVG\}$. However, we take into consideration only the first four attributes which refer to the range query. The first two attributes are the coordinates for the x-axis and y-axis for the center of the corresponding rectangle, respectively. The third and fourth attributes represent the ranges of the first two attributes. The D_w consists of 1.000 queries of the format $q_i = \{X_i, Y_i, Xr_i, Yr_i\}$. We randomly generate for each q_i a number $\omega_i \in [20{,}30]$ of two-dimensional data points which are located inside the rectangle of q_i. The total number of two-dimensional points is equal to 24.923 and constitutes the D_{2d} dataset. The last dataset D_T contains ψ=1.000 incoming Range queries with the same distribution and format with the D_w. Our goal is to confirm that the proposed model has the ability to detect the data that an incoming query into an SV needs. We use the D_w to 'train' the HCBM and HMCM model while the D_T dataset is used to test the performance of the models BM, HCBM and HMCM.

We evaluate the described models both for the error levels and the time that they need to detect the data. The evaluation of the models is relying on the metrics of Precision (PRE), Recall (REC), Accuracy (ACC), False Positive Rate (FPR) and F$_1$-score (FSC) as they ensue from the calculation of True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN). We define as TP the number of data that an incoming query q_i needs and they detected while TN the number of data that the q_i does not need and the model correctly reject them. On the other hand, FP is the number of data that the q_i does not need but the model retrieved them, and FN is the number of data that the q_i needs but the model does not detect them. The required time for each query is symbolized as τ_i .The formulas of Precision, Recall, FPR and F$_1$-score is defined as follows:

$$PRE = \frac{TP}{TP + FP} \tag{8}$$

[1] http://archive.ics.uci.edu/ml/datasets/Query+Analytics+Workloads+Dataset

$$REC = \frac{TP}{TP + FN} \tag{9}$$

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

$$FPR = \frac{FP}{FP + TN} \tag{11}$$

$$FSC = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{12}$$

The aforementioned metrics are used to calculate the performance of models for each q_i. However, the performance of the models has to be estimated over all the incoming queries, thus we use the average values of the previous metrics to calculate the overall performance of the models. Except from the metrics for the error calculation we also calculate the average time that every model needs to detect the data which satisfy the incoming queries. The overall metrics are defined as follows:

$$\mu_{PRE} = \frac{\sum_{q_i \in D_T} PRE_i}{\psi} \tag{13}$$

$$\mu_{REC} = \frac{\sum_{q_i \in D_T} REC_i}{\psi} \tag{14}$$

$$\mu_{ACC} = \frac{\sum_{q_i \in D_T} ACC_i}{\psi} \tag{15}$$

$$\mu_{FPR} = \frac{\sum_{q_i \in D_T} FPR_i}{\psi} \tag{16}$$

$$\mu_{FSC} = \frac{\sum_{q_i \in D_T} FSC_i}{\psi} \tag{17}$$

$$\mu_{Time} = \frac{\sum_{q_i \in D_T} \tau_i}{\psi} \tag{18}$$

The models have the best performance when the metrics $\mu_{Acc}, \mu_{PRE}, \mu_{REC}, \mu_{FSC}$ reach the unity and the metrics μ_{FPR} and μ_{Time} are closer to zero. In our experiments we pay great attention in the metrics μ_{FPR} and μ_{Time} because the former gives the average rate of data that the models retrieve but the incoming queries do not need them, while the latter one gives the average time that one of the methods needs to respond into the incoming queries. Figures 26-31, we present the performance of models for different number of clusters regardless of the way that they are created i.e., from the hierarchical clustering (HMCM) or the non-hierarchical clustering (HCBM). We have to mention that in the plots for the error metrics we do not include the BM model because it detects all the requirement data without error since it scans sequentially the entire databases.

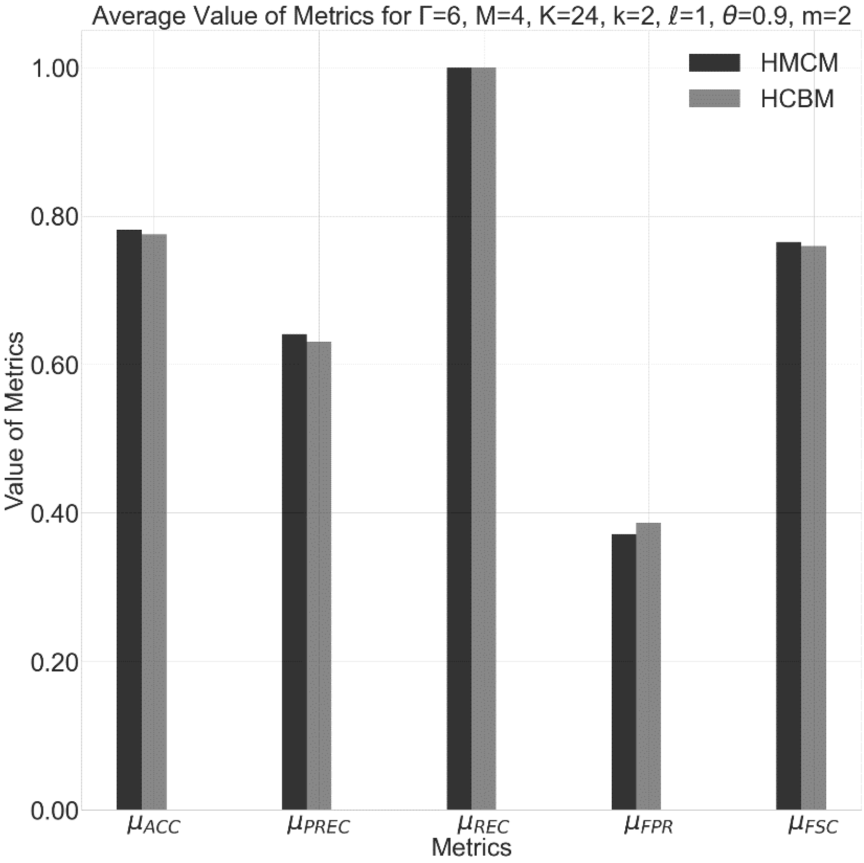


Figure 26: Comparison between HMCM and HCBM for Γ=6, M=4 and K=24

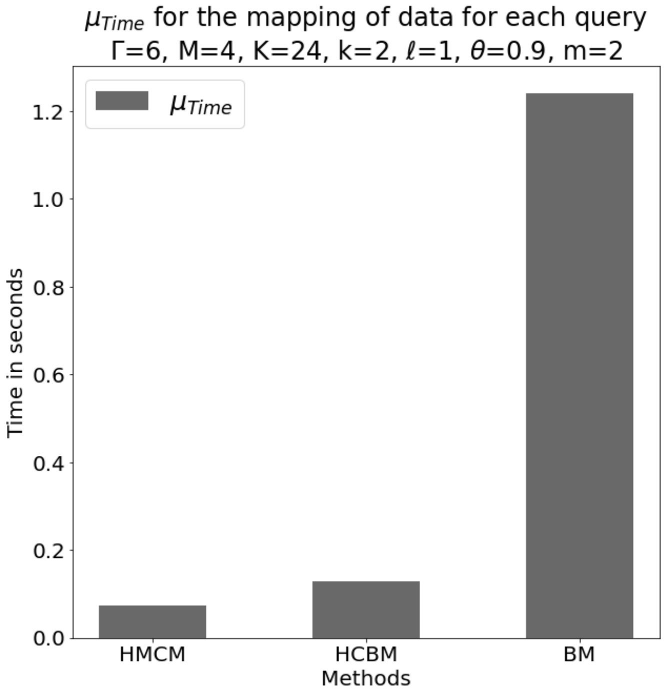


Figure 27: Average Time comparison between the models for Γ=6, M=4, K=24

In Figure 26 we compare the HMCM and HCBM when they created 24 clusters. As we can easily see, the HMCM has better performance for all metrics except the μ_{REC} where two models have the same performance. The dominance of the HMCM is revealed clearly from the Figure 27 where the average time that the HMCM needs to respond to the incoming queries is significant less from the BM method and much less from the HCBM model.

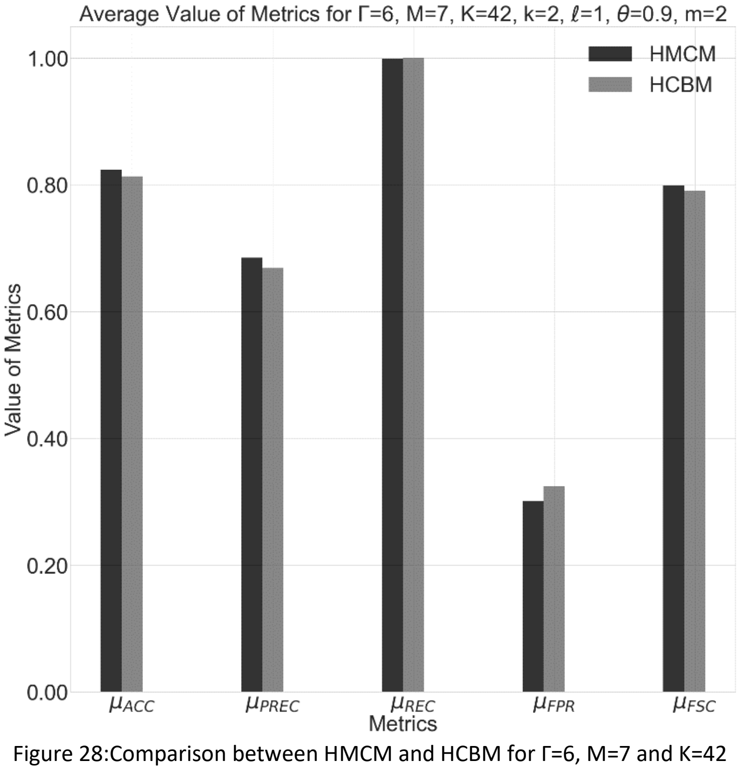


Figure 28:Comparison between HMCM and HCBM for Γ=6, M=7 and K=42

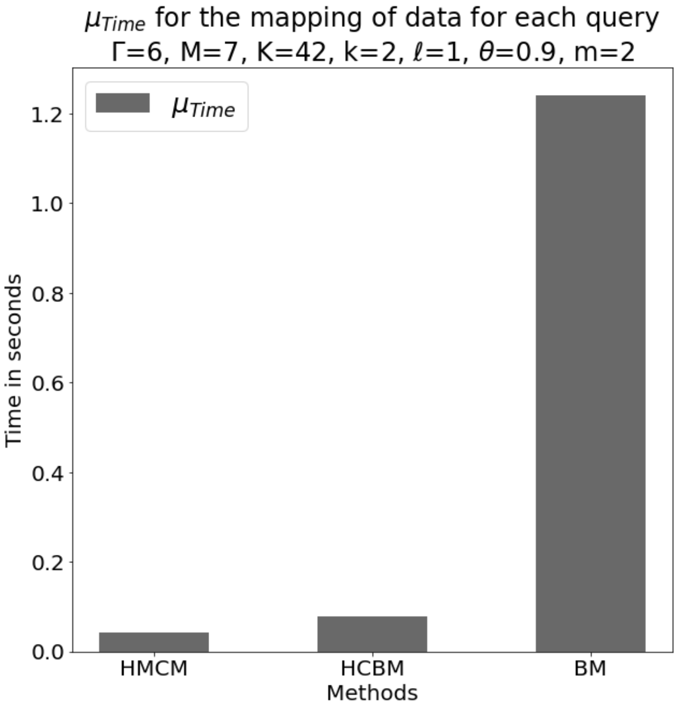


Figure 29:Average Time comparison between the models for Γ=6, M=7, K=42

Figure 28 shows the comparison of the HMCM and HCBM for 42 clusters. We observe that the HMCM overcomes the HCBM for all metrics except from the μ_{REC}, where the HMCM has slightly lower performance. Nevertheless, both μ_{FSC} and μ_{FPR} metrics confirm that the HMCM has better performance in error metrics. Figure 29 strengthens the conclusion that we deduce from Figure 28 since HMCM achieve better performance in less time.

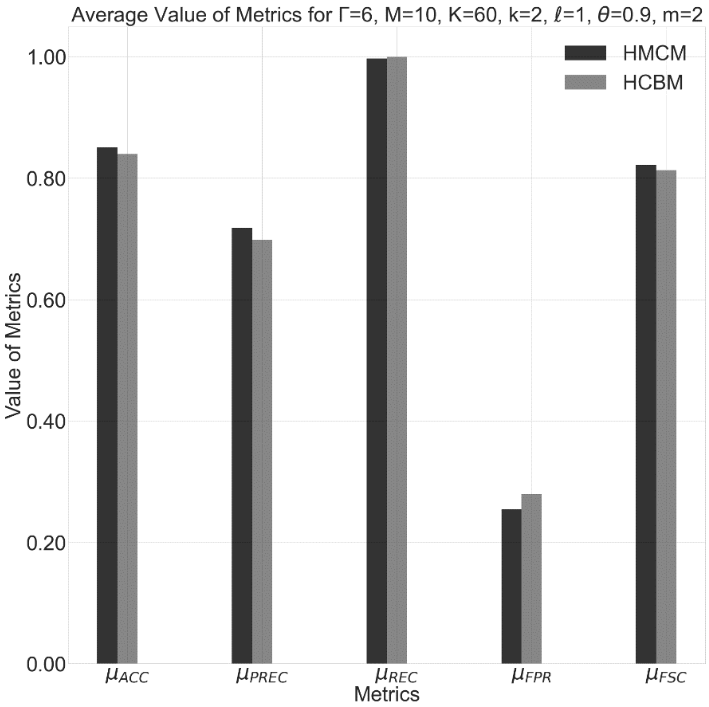


Figure 30:Comparison between HMCM and HCBM for Γ=6, M=10 and K=60

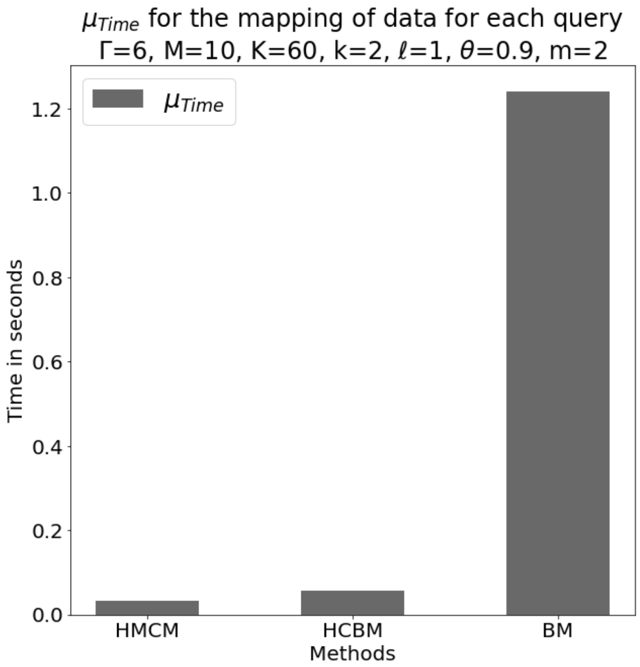


Figure 31:Average Time comparison between the models for Γ=6, M=10, K=60

In figures 30 & 31 we compare the error and time metrics between the models for 60 clusters respectively. In figure 30, the HMCM has better performance in the majority of the error metrics. As far as the time metric is concerned, the HMCM maps the data in less time that the other models, as it is presented in figure 31. Again, in the most important metrics for inferring the conclusion of the designation of the best model, the HMCM clearly outperforms the HCBM.

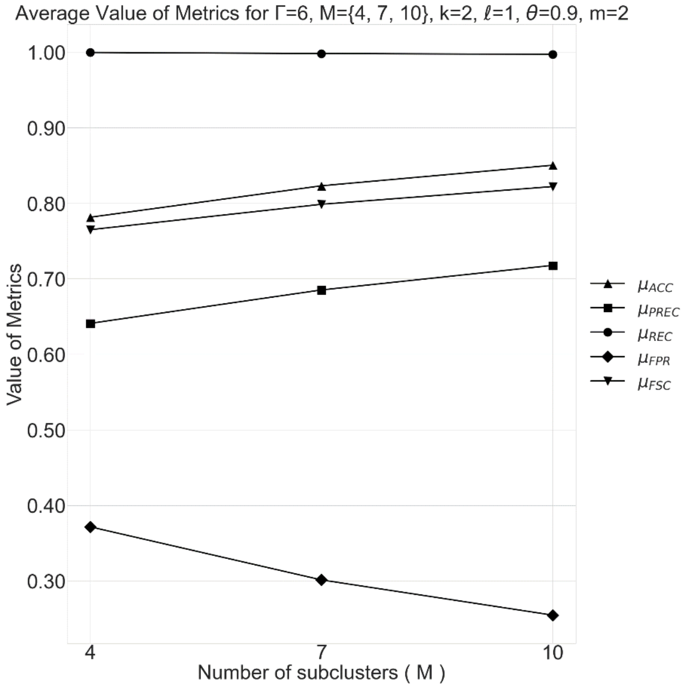


Figure 32: Performance of error metrics for different number of subclusters in HMCM

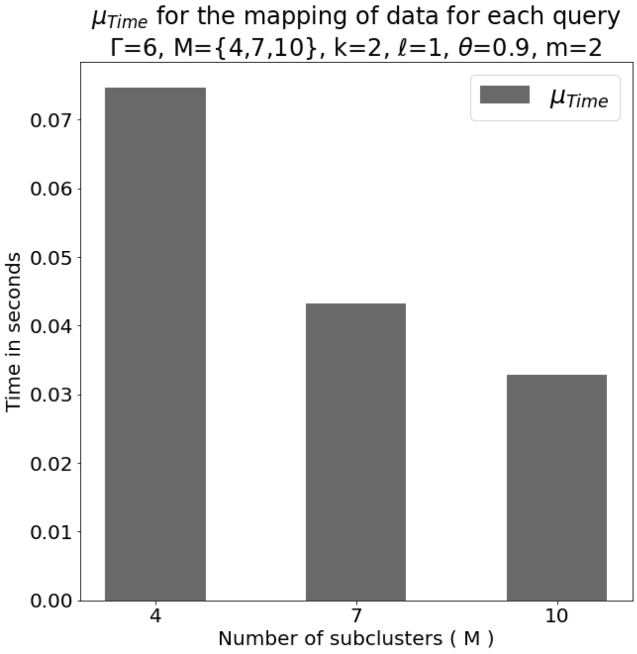


Figure 33: Comparison of the time metric for different number of subclusters in HMCM

In Figures 32 & 33, we evaluate the effect of the number of subclusters in the performance of our model both for error and time. We can easily observe that the increase of the number of subclusters clearly improves the performance of the HMCM model and simultaneously decrease the average required time for the mapping of data.

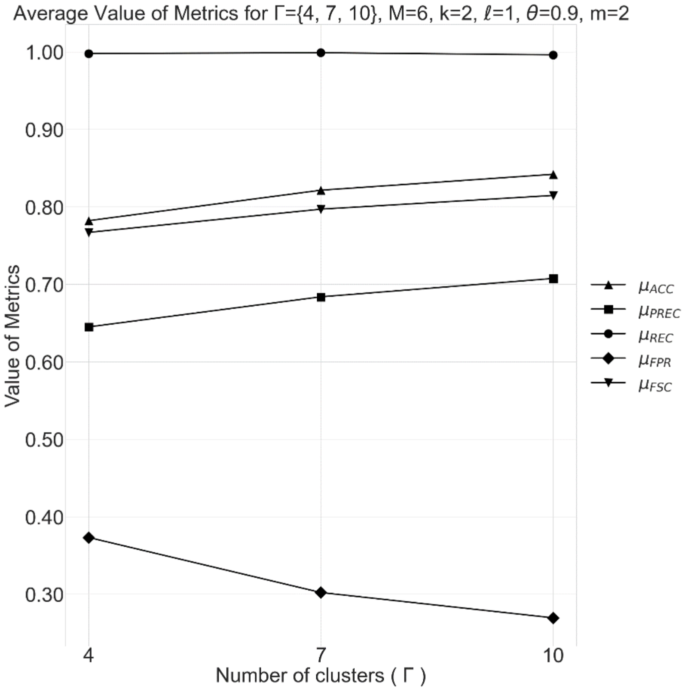


Figure 34:Performance metrics for different number of clusters in HMCM

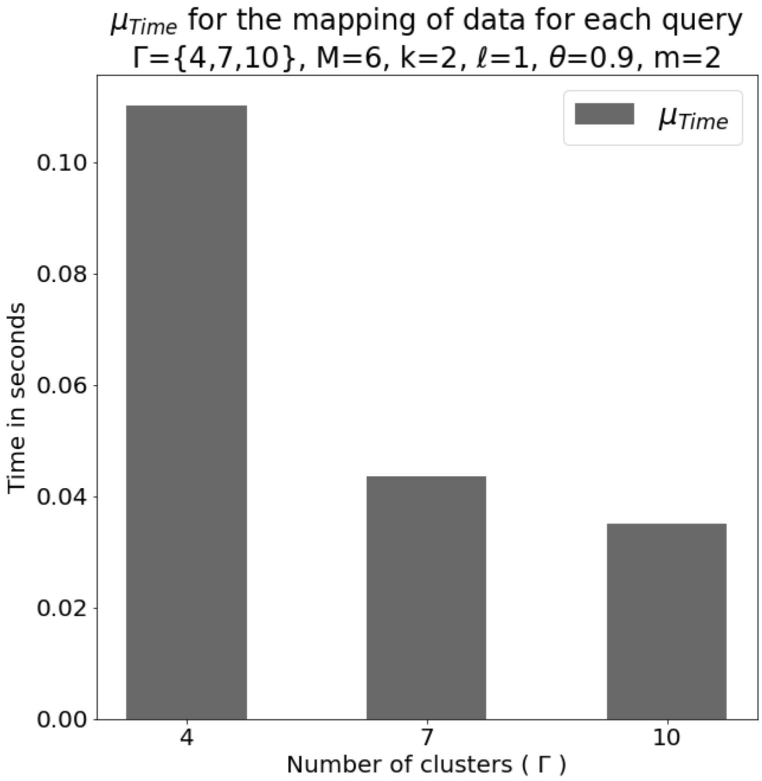


Figure 35:Comparison of the time metric for different number of clusters in HMCM

Figures 34 & 35 present the effect of the increase of clusters in error and time metrics for HMCM respectively. We notice that the higher the number of clusters the better the performance becomes, while the average time is affected in opposite direction.

Table 2: Comparison of same number of clusters with different combinations for the HCMC for $\theta = 0.9, k = 2, m = 2, \ell = 1$

Γ	**M**	$\boldsymbol{\mu_{ACC}}$	$\boldsymbol{\mu_{PRE}}$	$\boldsymbol{\mu_{REC}}$	$\boldsymbol{\mu_{FPR}}$	$\boldsymbol{\mu_{FSC}}$	$\boldsymbol{\mu_{Time}}$
6	10	0.850315099	0.71770102	0,99712312	0.254602569	0.822063888	0.032923581
10	6	0.84192673	0.707488132	0.996146484	0.269502654	0.814642176	0.035121117

From the previous figures we have to answer the question whether it is better to choose a higher number of clusters than subclusters or vice versa. Based on the previous experiments, we compare the performance of HMCM model for 60 clusters, using different combinations of the number of clusters and subclusters and present the results in table 2. As we can observe, choosing a higher number of subclusters improves the performance of HMCM model for all error metrics. At the same time, the required time for data mapping is decreased.

CHAPTER 9: CONCLUSIONS AND FUTURE WORK

Data mapping is a significant data management process which plays an important role in many applications domains. This process become more complex when the data are geo-distributed in different databases around the world. Data mapping can be improved if we can identify relations between the data in the DDBs and the queries that the users send to the server. In this thesis, we focus on the efficient mapping of data and propose a solution for an effective data mapping in minimum possible time. We are based on the answers of past queries, and propose an hierarchical clustering scheme which groups the past queries relying on the similarity of the requirements of the queries. Also, we involve a mechanism for the calculation of common interest data area between two queries. We adopt this type of strategy to create small groups of queries with similar data requirements and try to benefit from the exclusion of non-similar groups with an incoming query to reduce the time of response and to prevent the caching of unneeded data. We perform an extensive set of experiments to evaluate the proposed model in order to confirm its ability to map the data in short time with a small amount of extra unrequired data. A possible extension of this work could be the involvement of a more complex methodology for the improvement both of error and time metrics. Also, we could adopt a deep learning model that will be able to adapt our model to changes in user requirements expressed through queries.

REFERENCES

[1] W. Zhao, F. Rusu, B. Dong, K. Wu, A. Y. Ho and P. Nugent, "Distributed caching for processing raw arrays," *ACM International Conference Proceeding Series,* 2018.

[2] J. Wang, N. Ntarmos and P. Triantafillou, "GraphCache: A caching system for graph queries," *Advances in Database Technology - EDBT,* Vols. 2017-March, 2017.

[3] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden and I. Stoica, "BlinkDB: Queries with bounded errors and bounded response times on very large data," *Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys 2013,* 2013.

[4] K. Kolomvatsos and C. Anagnostopoulos, "A probabilistic model for assigning queries at the edge," vol. 102, pp. 865-892, 2020.

[5] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting and C. Binnig, "DeepDB: Learn from data, not from queries!," *Proceedings of the VLDB Endowment,* vol. 13, no. 7, 2020.

[6] K. Kolomvatsos and C. Anagnostopoulos, "Reinforcement learning for predictive analytics in smart cities," *Informatics,* vol. 4, no. 3, 2017.

[7] F. Savva, C. Anagnostopoulos and P. Triantafillou, "Adaptive learning of aggregate analytics under dynamic workloads," *Future Generation Computer Systems,* vol. 109, 2020.

[8] A. Silberschatz, H. F. Korth and S. Sudarshan, Database System Concepts (7th. edition), 2019.

[9] M. T. Özsu and P. Valduriez, Principles of Distributed Database Systems, 2020.

[10] S. K. Rahimi and F. S. Haug, Distributed Database Management Systems: A Practical Approach, 2010.

[11] P. Nerurkar, A. Shirke, M. Chandane and S. Bhirud, "Empirical Analysis of Data Clustering Algorithms," *Procedia Computer Science,* vol. 125, 2018.

[12] A. Nagpal, A. Jatain and D. Gaur, "Review based on data clustering algorithms," *2013 IEEE Conference on Information and Communication Technologies, ICT 2013,* 2013.

[13] M. Motwani, N. Arora and A. Gupta, "A study on initial centroids selection for partitional clustering algorithms," *Advances in Intelligent Systems and Computing,* vol. 731, 2019.

[14] E. Schubert and P. J. Rousseeuw, "Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),* vol. 11807 LNCS, 2019.

[15] D. Guo, J. Chen, Y. Chen and Z. Li, "LBIRCH: An improved BIRCH algorithm based on link," *ACM International Conference Proceeding Series,* 2018.

[16] G. Karypis, E. H. Han and V. Kumar, "Chameleon: Hierarchical clustering using dynamic modeling," *Computer,* vol. 32, no. 8, 1999.

[17] M. Li, X. Bi, L. Wang and X. Han, "A method of two-stage clustering learning based on improved DBSCAN and density peak algorithm," *Computer Communications,* vol. 167, 2021.

[18] Y. Wang, Y. Gu and J. Shun, "Theoretically-Efficient and Practical Parallel DBSCAN," *Proceedings of the ACM SIGMOD International Conference on Management of Data,* 2020.

[19] M. Hasan, J. Hanawa, R. Goto, H. Fukuda, Y. Kuno and Y. Kobayashi, "Person Tracking Using Ankle-Level LiDAR Based on Enhanced DBSCAN and OPTICS," *IEEJ Transactions on Electrical and Electronic Engineering,* vol. 16, no. 5, 2021.

[20] H. Hotait, X. Chiementin, M. S. Mouchaweh and L. Rasolofondraibe, "Monitoring of Ball Bearing Based on Improved Real-Time OPTICS Clustering," *Journal of Signal Processing Systems,* vol. 93, no. 2-3, 2021.

[21] M. Etehadtavakol, S. Sadri and E. Y. Ng, "Application of K- and fuzzy c-means for color segmentation of thermal infrared breast images," *Journal of Medical Systems,* vol. 34, no. 1, 2010.

[22] A. A. Jamel and B. Akay, "A survey and systematic categorization of parallel K-Means and fuzzy-C-Means algorithms," *Computer Systems Science and Engineering,* vol. 34, no. 5, 2019.

[23] N. Shi, X. Liu and Y. Guan, "Research on k-means clustering algorithm: An improved k-means clustering algorithm," *3rd International Symposium on Intelligent Information Technology and Security Informatics, IITSI 2010,* 2010.

[24] V. Schwämmle and O. N. Jensen, "A simple and fast method to determine the parameters for fuzzy c-means cluster analysis," *Bioinformatics,* vol. 26, no. 22, 2010.

[25] R. Gupta, S. K. Muttoo and S. K. Pal, "Fuzzy C-Means Clustering and Particle Swarm Optimization based scheme for Common Service Center location allocation," *Applied Intelligence,* vol. 47, no. 3, 2017.

[26] A. Stetco, X. J. Zeng and J. Keane, "Fuzzy C-means++: Fuzzy C-means with effective seeding initialization," *Expert Systems with Applications,* vol. 42, no. 21, 2015.