People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
8 May 1945 Guelma University

Faculty of Mathematics, Computer Science and Material Sciences
Department of Computer Science
Laboratory of Information and Communication Sciences and Technologies

THESIS

IN VIEW OF OBTAINING
THE DOCTORATE DEGREE IN 3rd CYCLE
Domain: Mathematics and Computer Science **Field:** Computer Science
Specialty: Computer Science Systems
Presented by:

Mrs. Karima KHETTABI

Entitled

Distributed Similarity Queries Search in Metric Space in IoT Systems

Defended: 23/03/2023 In front of the board of examiners composed of:

2022/2023

ملخص

مؤخرا الكمية الكبيرة للبيانات المستمرة ولغير متجانسة، المولدة من طرف مستشعرات وأجهزة أنترنت الأشياء (IoT)، جعلت من حفظ هده البيانات والبحث عن الاستعلام عملية صعبة جدا. معظم الطرق الموجودة في الأدبيات فشلت في معالجة متطلبات IoT .

في هذه الأطروحة، طريقة البحث kNN ممزوجة مع التوازي استعملت للبحث عن الاستعلام في بنيات مقترحة، مطورة في الفضاء المتري في هندسة ضباب-سحاب. الاقتراح الأول شجرة ثنائية مبنية على حاويات في مستوى حساب تجمعات الضباب-سحاب (شجرة B3CF) وهو فهرس بني لمزج التجميع بخوارزمية DBSCAN والتوازي. نتائج محاكاة بناء الشجرة والبحث عن الاستعلامات بطريقة kNN بالتوازي بين أن شجرة B3CF تعدت الفهارس الأخرى في الأدبيات وهدا ما جعل منها بديل قوي لفهرست بيانات IoT الكبير. الاقتراح الثاني هو طريقة معامل التغير (CV) التي طورت من أجل فهرست البيانات المستمرة، في هذه الطريقة التدفق الأول للبيانات جمع في مجموعات متجانسة باستعمال خوارزمية DBSCAN البيانات في هذه المجموعات فهرست مباشرة بالتوازي. بعد تجميع بيانات التدفق الآتي، البيانات في هده المجموعات إما تدخل في فهارس موجودة أو تبنى بها فهارس جديدة حسب قيمة معامل التغير (CV). هذه الطريقة أثبت فعاليتها من حيث بناء الفهارس والبحث عن الاستعلامات بطريقة kNN بالتوازي، وهذا بعد مقارنتها مع طريقتين تمثلان الحالتين الجديدتين وهما إنشاء فهرس جديد (CNI) وطريقة إدخال في فهرس موجود(IEI). الاقتراح الثالث هو طريقة المسافة العتبة (TD) والتي تشبه طريقة CV. لكن، في هذه الطريقة TD، المجموعات الآتية تفهرس أو تدخل في فهارس موجودة بالاعتماد على مقارنة المسافة بين مراكزها ومركز المجموعات الأولى مع مسافة حدية TD. هذه الطريقة تعدت طريقة إنشاء شجرة جديد (CNT) من حيث بناء الأشجاروالبحث عن الاستعلامات بطريقة kNN بالتوازي، لكن هي غير فعالة بعض

الشيء بمقارنتها مع طريقة CV. النتائج التجريبية بينت أن هاتان الطريقتان تعدتا بعض طرق الفهرسة في الأدبيات ويمكن اعتبارهما طرق بديلة لفهرست بيانات IoT المستمرة. الاقتراح الأخير

شجرة رباعية مبنية على حاويات في مستوى الحساب ضباب-سحاب (شجرة QCCF) التي فيها،البيانات تفهرس مباشرة دون تجميعها باستعمال خوارزمية DBSCAN مقارنة التجارب الخاصة ببناء الشجرة والبحث عن الاستعلام بطريقة kNN بالتوازي في عقد هذه الشجرة مع بض الفهارس في الأدبيات يبين أن شجرة QCCF هي أكثر فعالية من هاته الفهارس وهذا يجعل منها مرشحة لتكون طريقة بديلة لفهرست بيانات IoT ولو أنها أظهرت بعض الضعف أمام شجرة B3CF.

الكلمات المفتاحية

بيانات IoT الكبيرة؛ الفضاء المتري؛ البحث عن الاستعلام المتشابه؛ شجرة B3CF؛ التجميع؛ DBSCAN; التوازي؛ طريقة معامل التغير؛ طريقة المسافة العتبة؛ شجرة QCCF

Abstract

In recent years, the large amount of continuous and heterogeneous data generated by the Internet of Things (IoT) sensors and devices made their record and the query search tasks much more difficult. Most of the state-of-the-art methods have failed to deal with the new IoT requirements. In this thesis, the kNN search method combined parallelism was used for similarity queries search in proposed methods developed in metric space in the Fog-Cloud architecture. The first proposition is the Binary tree based on Containers at the Cloud-Clusters Fog computing level (B3CF-tree) which is an index constructed by combining DBSCAN clustering and parallelism. The simulation results of the index construction and the parallel kNN query search showed that the B3CF-tree surpassed those in literature. The second proposition is the Coefficient of Variation (CV) method which was developed for indexing continuous IoT data stream. In this method, the first data stream is grouped into clusters using the DBSCAN algorithm. Data in these clusters are directly indexed in parallel. After the clustering of the arrival data stream, the data in clusters are inserted in existing indexes or new indexes are constructed basing on the coefficient of variation value. This method has proven its efficiency in term of the indexes construction and the parallel kNN query search compared with two other methods representing the two utmost cases namely the Creation of a New Index (CNI) and the Insertion in an Existing Index (IEI) methods. The third proposition is the Threshold Distance (TD) method which looked like the CV method. However, in the TD method, the arrival clusters are indexed or inserted in existing indexes basing on the comparison of the distance between their centers and the first clusters centers with a threshold distance TD. This method outperforms the Creation of a New Tree (CNT) method in terms of trees construction and parallel kNN search however, it is quite insufficient compared with the results of the CV method. The experimental results showed that Both methods surpassed some indexing methods in literature and could be considered as an alternative method for indexing continuous IoT big data. The last proposition is the Quad tree based on Containers at the Cloud- Fog computing level (QCCF-tree) in which data are directly indexed without clustering. The comparison of the experimental results of the index construction and the parallel kNN search in the index nodes with some indexes in literature showed that the QCCF-tree is more efficient than these indexes. This made it a candidate as an alternative method for big IoT data indexing even though it presented a weakness in front of the B3CF-tree.

Keywords

Big IoT data; Metric space; Similarity queries search; B3CF-tree; Clustering; DBSCAN; Parallelism; Coefficient of Variation method; Threshold Distance method; QCCF-tree

Résumé

Récemment, la grande quantité de données continues et hétérogènes, générées par les capteurs et les composants IoT, a rendu l'enregistrement des données et la recherche des requêtes des taches très difficiles. La plus part des méthodes de l'état de l'art ont échoué de traiter les exigences de l'IoT. Dans cette thèse, la méthode de recherche kNN combinée avec le parallélisme a été utilisée pour la recherche des requêtes similaires dans des structures proposées développées dans l'espace métrique dans l'architecture Fog-Cloud. La première proposition est le Binary tree based on Containers at the Cloud-Clusters Fog computing level (arbre-B3CF) qui est un indexe construit par la combinaison du regroupement par l'algorithme DBSCAN et le parallélisme. Les résultats de simulation de la construction de l'arbre et de la recherche des requêtes par la méthode kNN parallèle ont montré que l'arbre-B3CF a dépassé les autres dans la littérature qui fait de lui un alternatif fort pour l'indexation des grandes données IoT. La seconde proposition est la méthode de Coefficient of Variation (CV) qui a été développée pour indexer les données continues. Dans cette méthode, le premier flux de données est groupé dans des clusters en utilisant l'algorithme DBSCAN. Les données dans ces clusters ont été directement indexées parallèlement. Après le regroupement des données du flux arrivant, les données dans ces clusters sont insérées dans des indexes existants ou de nouveaux indexes seront construits selon la valeur du coefficient de variation. Cette méthode a prouvé son efficacité, en terme de la construction des indexes et la recherche des requêtes par la méthode kNN parallèle, en la comparant avec deux méthodes représentants les deux cas extrêmes notamment la méthode Creation of a New Index

(CNI) et la méthode Insertion in an Existing Index (IEI). La troisième proposition est la méthode Threshold Distance (TD) qui ressemble à la méthode CV. Cependant, dans la méthode TD, les clusters arrivants sont indexés ou insérés dans des indexes existants en se basant sur la comparaison de la distance entre leurs centres et ceux des premiers clusters avec une distance seuil TD. Cette méthode a surpassé la méthode Creation of a New Tree (CNT) en termes de la construction des arbres et la recherche kNN parallèle cependant, elle est un peu inefficace en la comparant avec la méthode CV. Les résultats expérimentaux ont montrés que ces deux méthodes surpassées quelques méthodes d'indexation dans la littérature et peuvent être considérées comme des méthodes alternatives pour l'indexation des données IoT continues. La dernière proposition est le Quad tree based on Containers at the Cloud- Fog computing level (arbre-QCCF) dans laquelle, les données sont directement indexées sans le regroupement par l'algorithme DBSCAN. La comparaison des résultats expérimentaux de la construction de l'arbre et de la recherche kNN parallèle dans les nœuds de l'arbre avec quelques indexes dans la littérature a montré que l'arbre-QCCF est plus efficace que ces indexes. Cela fait de lui un candidat comme une méthode alternative pour l'indexation des données IoT même s'il présente une faiblesse face à l'arbre-B3CF.

Mots clés

Données massives d'IdO; Espace métrique; Recherche de requêtes de similarité; Arbre-B3CF; Clustering; DBSCAN; Parallelisation; Méthod du Coefficient de Variation; Méthode de la Distance Seuil; Arbre-QCCF

Contents

List of Figures

List of Tables

List of Abbreviations

B3CF Binary tree based on containers at the cloud-clusters fog computing level. 131

BCCF Binary tree based on containers at the cloud-fog computing level. 119

BH Binary tree with Hyper-plane. 168

CNI Creation of a New Index. 170

CV Coefficient of Variation. 164

DBSCAN Density-Based Spatial Clustering of Applications with Noise. 62

GH Generalised Hyper-plane tree. 97

IEI Insertion in an Existing Index. 170

IoT Internet of Things. 21

kNN k Nearest Neighbor. 71

L2H Learning to Hash methods. 74

LSH locality sensitive hashing methods. 73

MBR Minimum Bounding Rectangles. 81

NIST National Institute of Standards and Technology. 32

QCCF Quad-tree based on Containers at the Cloud-Fog computing level. 208

RFID Radio Frequency IDentification. 22

TD Threshold Distance. 193

WSN Wireless Sensor Network. 29

Introduction

Motivations

In the last decades, the Internet of Things (IoT) has found a wide range of use such as in smart cities, in smart home and in health care. This technology supports a large number of physical objects and devices with identities, personalities and network capabilities (Things) to transparently communicate and interact between them and with other network resources (Internet). These IoT devices provide services to facilitate life. They are heterogeneous and in many cases, they are deployed in distributed and dynamic environments over a large geographic region. They generate huge data that can overwhelm storage systems and causes a serious increase in their recovery time. A new forecast from International Data Corporation (IDC) estimates that there will be 41.6 billion connected IoT devices, or "things," generating 79.4 Zetta bytes (ZB) of data in 2025 [1]. The problem of data latency is considered as a serious obstacle when using cloud computing for storage and process of this big IoT data. The causes of this data delay are still obvious [2]. Several researches have been made to address big IoT data storage and various papers have been published [3], [4] to improve cloud computing of data storage and queries retrieve algorithms. Recently, a few researches have addressed the data storage and retrieve using the fog computing [5], [6] because of its interesting characteristics such as the closeness to the end users and the computation capabilities. In addition, in the fog computing, big IoT data could be distributed in many fogs located in different geographic regions. Benefiting from the fog characteristics in indexing big IoT data will improve considerably the similarity queries search. Indeed, indexing of large-scale IoT data must be efficient, dynamic and support different data types.

Context of the Study

The massive data, generated by interconnected IoT devices require storage, process, analysis and finding effective methods for similarity queries search. To store these big IoT data, indexing methods are used. Indexing is one of the most widely used mechanisms to provide rapid access to data. Indexing is a data organization step that must allow efficient access to the data efficiently when performing similarity queries. The principle is to organize similar data to speed up searches [7]. The goal of any index is therefore to provide fast access to the objects in a database, by reducing the search space, the cost of input/output and the number of calculations of distances between objects. In other words, the index provides the efficient implementation of associative search [8]. For a better management of big IoT data, the fog computing architectures are currently used. The hierarchical fog architecture consists of three layers: terminal layer, fog layer and cloud layer [9]. Indexing of large-scale IoT data must be efficient, dynamic and support different data types. Metric spaces became popular in the indexing process. In order to exploit not the data representation itself, which has become too rich and complex, but to work "only" on the similarities that can be computed between objects.

Furthermore, the nature of big IoT data is dynamic and its underlying data distribution can change over time. Another point is that the data is produced in real time. This necessitates development of IoT specific data analytics solutions which can handle the heterogeneity, dynamicity and velocity of the data. To group the data coming from the devices, clustering methods are used in which, the data is usually clustered according to different criteria; e.g. similarity and homogeneity. The clustering results in a data analysis scenario can be interpreted as categories in a dataset and can be used to assign data to various groups i.e. clusters [10]. The grouping of IoT data into clusters may allow the introduction of parallelism during both the indexes construction and similarity queries search.

Objectives and Contributions

IoT systems are comprised of various devices that generate heterogeneous IoT data continuously. This continuity involved a big challenge concerning the data indexing and the query search in the dynamic IoT environment. The traditional indexing methods became inadequate to index the big IoT data because they suffer from the issue of the degradation in large scale and they are unable to extend with the permanent collection of data. In addition, the direct use of the cloud infrastructure affected negatively the communication time due to the big physical distances between the data sources and the data warehouse. The aim of this thesis is to propose new systems for indexing and retrieving data in an IoT environment that allows dealing with the index degradation and network congestion while ensuring minimal search time with optimal results quality. To reach our objective, the following tasks were addressed:

Proposition of a novel taxonomy This taxonomy is based on the grouping of indexes of different types of data into centralized methods and distributed methods.

Relocalization of the indexing process from the cloud to the fog nodes In order to bring the data as close as possible to the indexing structure in order to considerably reduce network congestion. In addition, each fog node generates the indexing structure of the distributed IoT data which not only allows parallelism during the construction of trees, but also allows it during the queries search process through the simultaneous launch of the same query on all fog nodes.

Division of the fog layer into levels In the cloud-fog architecture, the fog layer is divided into several levels in order to make a multi-steps indexing process.

Use of metric space The indexes constructed in multidimensional space suffer from the depending on a specific data type and dimensions. The data processing, in the metric space, is easier to index because it depends only on the distance between objects whatever their types.

Data clustering as a first step indexing process The DBSCAN algorithm is used, in the first fog level, to group IoT data into homogeneous clusters in order to reduce the data overlapping and index degradation.

Parallel construction of trees This process takes place in the second fog level. In this level, B3CF-trees (Binary tree based on containers at the cloud-clusters fog computing level) of clusters resulting from the use of the DBSCAN algorithm in the clustering fog level, were constructed simultaneously.

Use of hyper-planes for space partitioning For indexing IoT data in clusters, B3CF-trees are based on the metric space partitioning into hyper-planes using two pivots in order to guarantee a no-overlapping in indexes.

Indexing continuous data stream using the Coefficient of Variation (CV) method In order to index continuous IoT data stream in BH-trees (Binary trees with Hyper-plane), the Coefficient of Variation (CV) method is used in the cluster processing fog level located between the clustering level and the indexing level.

Indexing continuous data stream using the Threshold Distance (TD) method In the Threshold Distance (TD) method, proposed for indexing continuous IoT data stream in GHTs (Generalised Hyper-plane Trees), the fog layer is divided only into a clustering level and an indexing level.

Use of balls for space partitioning A proposed index called QCCF-tree (Quad-tree based on Containers at the Cloud-Fog computing level) is constructed, in the fog node, basing on the metric space partitioning into four balls using four pivots in order to reduce the index degradation and to speed up the query search. In this approach, the fog layer is not divided.

Use of parallel kNN search in the proposed binary trees The use of the DBSCAN algorithm for data clustering allows not only the parallel construction but also the parallel query search. The kNN search method is combined with parallelism in order to improve the similarity queries search process.

Use of parallel kNN search in the QCCF-tree nodes The parallelism is combined with the kNN search method in the inner of the QCCF-tree i.e. in the QCCF-tree nodes, in order to speed up the similarity queries search.

Overview of the Thesis

This thesis will be presented in two parts in addition to the introduction and the conclusion sections. The first part, untitled IoT data indexing in metric spaces: definitions and related work, contains four chapters. The first chapter deals with the mathematical definitions of some concepts and the different methods of similarity query search in metric space. In the second chapter, the internet of things definition will be presented as well as the characteristics and the different challenges in addition to an overview in the cloud computing and the fog computing. In the third chapter, big IoT data is defined and the clustering methods are described in detail in addition to other data analytics methods. The fourth chapter regrouped a state of the art concerning the centralized and the distributed indexing methods in multi-dimensional and in metric spaces by focusing on their advantages and limitations. The proposed approaches, in metric space, in the cloud-fog architecture are gathered in the second part which contains four chapters. The first chapter presents the parallel kNN search in the proposed B3Cf-trees constructed using the DBSCAN clustering combined with parallel indexation. In the second chapter, the parallel kNN search is used for similarity search in BH-tree constructed for indexing continuous data stream using the Coefficient of Variation (CV) method. The third chapter presents the parallel kNN search of queries in GH-trees constructed for indexing continuous IoT data stream using the Threshold Distance (TD) method. The last chapter presents the parallel kNN query search in the QCCF-tree nodes constructed by the indexing of the whole IoT data in a quad tree. For each proposition, a detailed description, algorithms, a description of the computation platform, the used datasets, the computation parameters are provided. The experimental results in terms of index construction and the kNN query search process will be presented, discussed and compared with those in literature for each proposition. A comparison between the proposed approaches will be also provided.

Part I

IoT data Indexing in Metric Space: Definitions and Related Work

1 Metric Spaces

1.1 Introduction

The widespread use of smart objects connected to Internet such as sensors, actuators and embedded devices, has led to an increase in the amount of collected data [11]. The type of these data is heterogeneous, dynamic and its corresponding data distribution can change over time [10]. On the other hand, the data comes in large quantities and is produced in real-time. This data needs to be processed and stored in a manner allowing its retrieve quickly. Many approaches, developed in the the multidimensional space, have presented some disadvantages when storing this heterogeneous data in terms of size, type and dimension [12], [13], [14]. In this work, the metric space is proposed to be the right compromise since, in this space, only distances between data are used regardless of their types and dimensions [15]. Metric space has been proposed before as a universal abstraction for data [16]. Furthermore, multidimensional spaces is a special cases of metric spaces. In the vector space, objects are represented by vectors and geometric properties of some of these vectors can be used for research. These characteristics, of course, cannot be extended to metric distances [17].

In this chapter, we present the mathematical definition of metric space and the ball and the hyperplane data partitioning concepts in metric space. Lastly, we provide definitions of some similarity query search methods in metric space.

1.2 Metric Space Definition

A metric space is a set of objects where a notion of distance between objects is defined. The mathematical definition is given as follows:

Definition 1.2.1 (Metric space)**.** A metric space $\mathcal{M} = (\mathcal{O}, d)$ is defined by a distance function d and a dataset $\mathcal{O}$. The distance function d measures the similarity between two elements from the given dataset $\mathcal{O}$. Similar objects correspond to smaller distances. Being a metric space $(\mathcal{O}, d)$ where $\mathcal{O}$ a set of points and d a distance function defined as: $d : \mathcal{O} \times \mathcal{O} \to \mathbb{R}^+$. The distance function d characterized by:

$$1.Non-negativity : \forall(x, y) \in \mathcal{O}^2, d(x, y) \geqslant 0. \tag{1.1}$$

$$2.Reflexivity : \forall x \in \mathcal{O}, d(x, x) = 0. \tag{1.2}$$

$$3.Symmetry : \forall(x, y) \in \mathcal{O}^2, d(x, y) = d(y, x). \tag{1.3}$$

$$4.Triangle-inequality : \forall(x, y, z) \in \mathcal{O}^3, d(x, y) + d(y, z) \leqslant d(x, z). \tag{1.4}$$

1.3 Multidimensional Space Definition

A multidimensional space is defined as a set of objects, called vectors, homogeneous or heterogeneous.The most usual case is that of orthonormal subspaces, i.e. defined on $\mathbb{R}^+$. In a vector space, objects are represented by vectors and the geometrical properties of these vectors are exploited for research. However, these properties cannot be extended to metric spaces [17]. Multidimensional spaces are subsets of the metric space, so any norm on a multidimensional space is a subset of a metric space [18].

1.4 Distance Functions

Distance functions are tools for measuring the proximity between different objects and are suitable for specific applications. These metrics are based on coordinates and can be divided into two groups: discrete distance functions and continuous distance functions. Discrete distance functions give only a small set of values, whereas with continuous distance functions, the cardinality of the resulting set of values is very large or infinite.

Furthermore, distance functions may be classified in terms of their cost of calculation, just by taking into account their approximate complexity.

1.4.1 Minkowski distances

The Minkowski distances are a family of metric functions, known as L_p metrics, because the individual cases depend on the numerical parameter p which vary according to the type of data. The function is defined on n-dimensional vectors of real numbers which can be transformed into vectors of numbers. The mathematical definition is given as follows:

Definition 1.4.1 (Minkowski distances)**.** The Minkowski distances is defined by two vectors $X \in \mathbb{R}^n = (x_1, x_2, \cdots, x_n)$ and $Y \in \mathbb{R}^n = (y_1, y_2, \cdots, y_n)$, with p is an integer. we define the Minkowski distances where,

$$L_p[X, Y] = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{1.5}$$

In the Minkowski distances, the most used values of the parameter p are $p = 1, p =$

2 and $p = \infty$ (Figure 1.1). Each curb represents a set of points, in the plane, at the same distance from the central point. A set of points, in the plane, at the same distance from the central point. $p = 1$ is translated by a losange, the circle for the metric $p = 2$ and $p = \infty$ results in a square. The intermediate values produce a progressive bulge from the lozenge to the square via the circle [18].

1. For $p = 1$: usually the Manhattan distance, which is expressed in the following equation:

$$L_1[X,Y] = \sqrt{\sum_{i=1}^{n} |x_i - y_i|} \tag{1.6}$$

2. For $p = 2$: distance indicates the Euclidean distance, the equation is:

$$L_2[X,Y] = \sqrt[2]{\sum_{i=1}^{n} |x_i - y_i|^2} \tag{1.7}$$

3. For $p = \infty$: known as the Chebyshev distance, the maximum distance and the infinite distance. Its equation is:

$$L_\infty[X,Y] = \max_{i=1}^{n} |x_i - y_i| \tag{1.8}$$

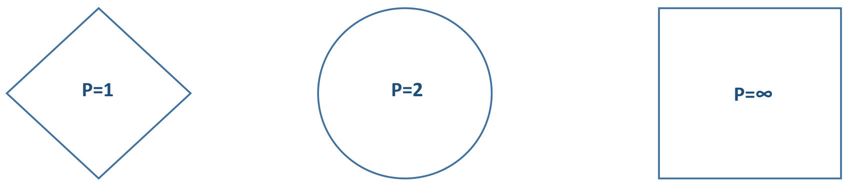


Figure 1.1: Different L_p distance functions [15].

1.5 Concepts of Ball and Hyperplane

The partitioning, in large, represents the basic principles of all storage structures, designed to partition the search space into subsets, such that once a query is answered, only certain of these subsets will be searched. For partition in a metric space, the data set does not possess coordinates that can be used in the geometric divisions. To solve this issue the metric space is based on selecting an object and promoting it to the pivot. All the other objects are classified by calculating the distance from this pivot. The choice of a certain value of distance acts as a threshold value and partitions the objects into two subsets. The researchers often exploit two key concepts: the ball partition and the hyperplane partition.

1.5.1 Ball partitioning

A ball is a general concept that allows us to generalize the disc in the Euclidean plane and the sphere in space. It is a set $\mathcal{O}$ of a metric space defined by a center object, or "pivot p", and a radius r (Figure 1.2). The use of pivot $p \in \mathcal{O}$ and radius $r \in \mathbb{R}^+$ allows the division of objects into two subsets S_1 and S_2 . The formal definition the ball partitioning is given:

Definition 1.5.1 (Ball)**.** Let $\mathcal{M} = (\mathcal{O}, d)$ be a metric space. Let $p \in \mathcal{O}$ be a pivot object and $r \in \mathbb{R}^+$ the covering radius. Then $Ball(\mathcal{O}, d, p, r)$ that is $Ball(p, r)$, where there is no ambiguity in the metric space defined a ball which partitions the space into two subsets S_1 and S_2:

$$\begin{aligned} S_1 &= \{o \in \mathcal{O}, d(p, o) \leqslant r\} \\ S_2 &= \{o \in \mathcal{O}, d(p, o) > r\} \end{aligned} \tag{1.9}$$

The redundant conditions $\leqslant$ and $>$ provide balance when the pivot is a median

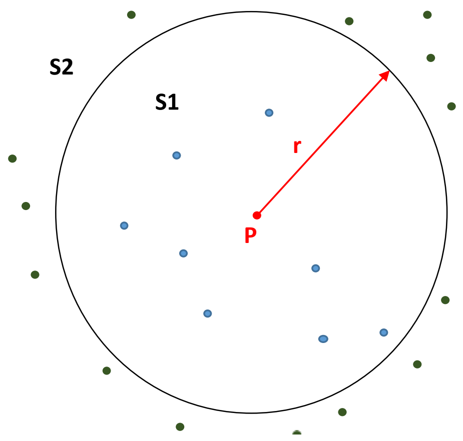


Figure 1.2: Ball partitioning scheme [18].

value and is not unique. This is achieved by affecting each element at the median distance to one of the subsets in an arbitrary, however balanced, method.

1.5.2 Hyperplane partitioning

This partitioning concept also splits the set $\mathcal{O}$ into subsets by two pivots p_1 and p_2, which are chosen arbitrarily (Figure 1.3). The rest of the objects o is assigned to S_1 or S_2 according to their distances to the selected pivots p_1 and p_2 as follows:

Definition 1.5.2 (Generalized hyperplane partitioning)**.** Let $\mathcal{M} = (\mathcal{O}, d)$ be a metric space. Let $(p_1, p_2) \in \mathcal{O}^2$ be two pivots object with $d(p_1, p_2) > 0$. Then $H(\mathcal{O}, d, p_1, p_2)$ that is $H(p_1, p_2)$, where there is no ambiguity in the metric space defined a hyperplane which partitions the space into two subsets S_1 and S_2:

$$\begin{aligned} S_1 &= \{o \in \mathcal{O}, d(p_1, o) \leqslant d(p_2, o)\} \\ S_2 &= \{o \in \mathcal{O}, d(p_1, o) > d(p_2, o)\} \end{aligned} \tag{1.10}$$

The generalized hyperplane eliminate the overlapping between the data. Unlike the ball partitioning, the generalized hyperplane is not able to assure a balanced distribution and a well adapted selection of pivot to attain this result is an attractive problem.

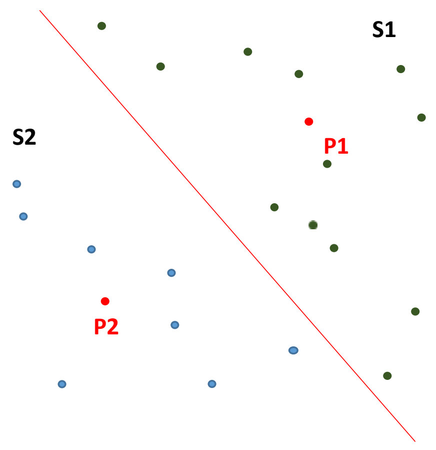


Figure 1.3: Hyperplane partitioning scheme [18].

1.6 Similarity Query Search

The intense use of IoT devices induced the emergence of unstructured IoT data that contains many types of data such as images, videos and time series. These types of data cannot be organized in a classical way or searched in a very significant way using accurate database queries which would retrieve exact results. The more common approach to similarity search, enabling always the building of index structures, to be exploited in different spaces. Similarity search is a manner of information retrieval in that the query is an example of an object and the result desired is a set of objects considered similar - in some sense - to the query [19]. A similarity query is given as an explicit or an implicit definition of a query object q and by using a constraint on the form and range of the neighborhood query. The resulting response to a query finds any objects that satisfy the constraint, which are guaranteed to be the objects that are near the given query object.

1.6.1 Range query method

A range query $R(q, r)$ finds all objects o within a set $X \subseteq \mathcal{O}$ that have a distance, from the query q, less than r (Figure 1.4).

$$R(q, r) = \{o \in X, d(o, q) \leqslant r\} \tag{1.11}$$

In the range query the query object q do not necessarily have to exist in the set $(X \subseteq \mathcal{O})$ to be searched [18].

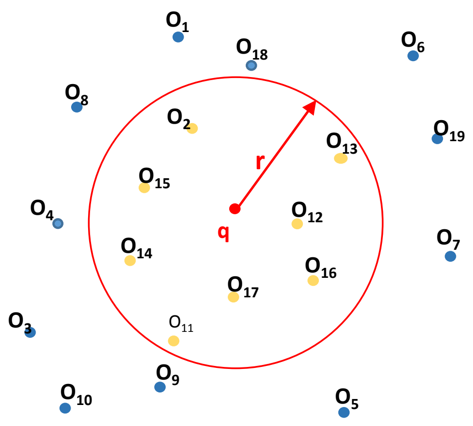


Figure 1.4: Range query description.

1.6.2 Similarity join method

The similarity join is performed by two different sets $X \subseteq \mathcal{O}$ and $Y \subseteq \mathcal{O}$. It was created by the need to use unstructured data to provide structured services [15]. The similarity join of two dissimilar datasets $(X \subseteq \mathcal{O})$ and $(Y \subseteq \mathcal{O})$ retrieves all object pairs $(x, y) \in (X, Y)$ with distance not greater than a threshold value

$\mu \geq 0$. The similarity join method is formally defined by the following equation:

$$J(X, Y, \mu) = \{(x, y) \in (X, Y), d(x, y) \leqslant \mu\} \tag{1.12}$$

The dataset X may be identical to the dataset Y, this case is called the self similarity join (Figure 1.5). If $\mu = 0$, we get the traditional natural join.

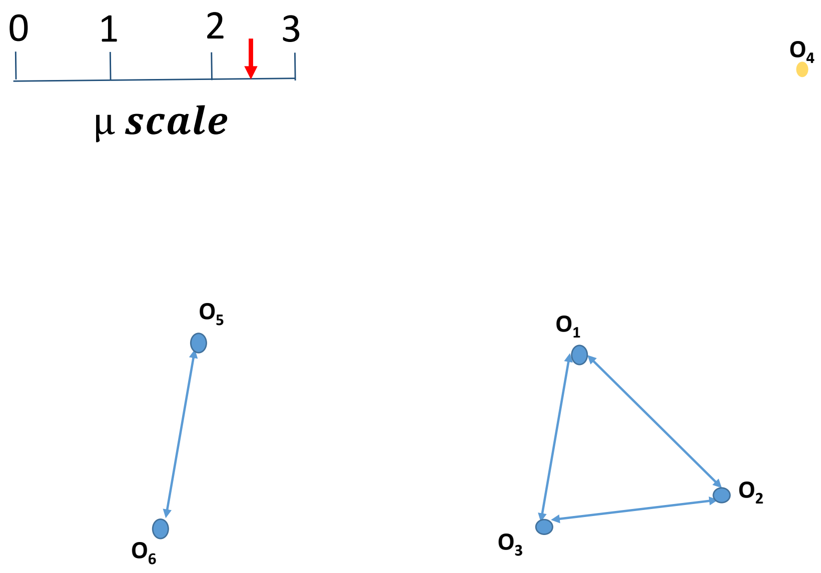


Figure 1.5: Similarity self join query with $\mu = 2.5$ [15].

1.6.3 Reverse nearest neighbor query method

Inverting the nearest neighbor query, finds the objects in the set closest to the query object q. The objects see the query object q as their nearest neighbor. It's referred to as a reverse nearest neighbor search [15]. The basic definition of this query search method is to find every object related with q as a k nearest neighbor (Figure 1.6). In this figure, dotted circles represent the distances to the second closest neighbor of the objects O_i. The objects o_4 and o_5 satisfy the query 2RNN(q), i.e. objects with q among their two nearest neighbors are represented by blue dots [15]. The response set of the general query kRNN(q) is given by the following relation [15]:

$$kRNN(q) = \{S \subseteq X, |S| = k, \forall x \in S : q \in kNN(x) \wedge \forall x \in X - S : q \notin kNN(x)\} \tag{1.13}$$

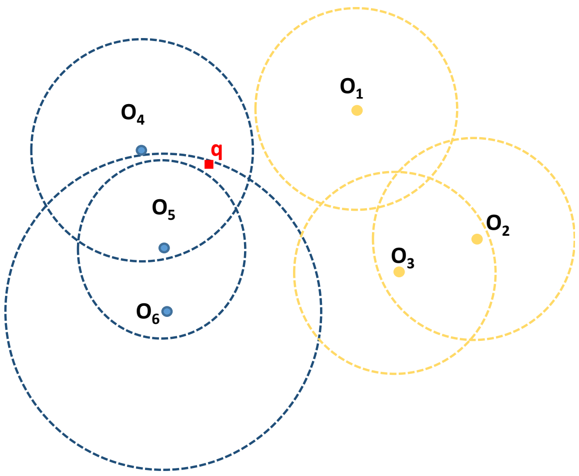


Figure 1.6: Reverse nearest neighbor query with k=2 [15].

1.6.4 Nearest Neighbor query method

The basic definition of this query search method is to find the closest object to the given query object q, i.e. the nearest neighbor of q [18]. The general case is where we search the k nearest neighbors (kNN). Specifically, kNN query finds the k nearest neighbors of the object q. Figure 1.7 illustrates the situation for $k = 5$ the objects O_4, O_{10}, O_5, O_{19} and O_{18} are closest to the query q. Formally, the set of the responses is defined as follows [15]:

$$kNN(q) = \{S \subseteq X, |S| = k \wedge \forall x \in S, y \in X - S : d(q,x) \leqslant d(q,y)\} \tag{1.14}$$

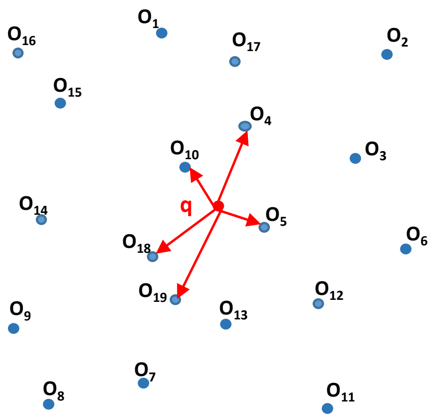


Figure 1.7: kNN query search with k=5.

1.7 Conclusion

The proposition of processing heterogeneous data, in term of type and dimension, in metric space as an alternative of the multidimensional space will be of a great interest. Indeed, in metric space, from the distance values, it is possible to distinguish objects of high dimension namely, in the case of data that does not follow uniform distribution which is the case of the whole real data such as Internet of Things (IoT) data. Using ball or hyperplane data partitioning in metric space makes easy the distinction between the right objects and the dismissed objects during the use of a similarity query search methods such as kNN.

2 Internet of Things (IoT)

2.1 Introduction

Nowadays, all devices such as smart home, smartphones, healthcare ones and home appliances have been applied for data generating. These massive data, generated by these interconnected devices, are known as Internet of Things (IoT). It is a dynamic network infrastructure, where physical and virtual "objects" have identities, physical attributes, virtual personalities, and intelligent interfaces. In the past few years, numerous researches have been realized on IoT [20],[21]. However, very few publications have been found discussing and pointing out the challenges of IoT [22]. The aim of this work is to address one of these challenges which is the storage and the retrieve of information.

In this chapter, we present the definition and the application of IoT before presenting IoT challenges. The cloud computing and the fog computing, proposed as solutions of IoT challenges are described in the end of this chapter.

2.2 Ubiquitous Computing in the Future Decade

The development of technology has caused the transition from the stage of personal computers to smartphones and other portable devices and the interaction between them has changed our daily lives which induced a fundamental transformation in computing called Ubiquitous computing (UbiComp) [23]. Several approaches to ubiquitous computing have been appeared in the literature [24], [25], [26]. The one is the Weiser's Calm Computing approach which was proposed by Mark Weiser, the ancestor of Ubiquitous computing [24]. He defined the intelligent environment as "the physical world richly and invisibly interwoven with sensors, actuators, displays and computational elements, seamlessly integrated into the everyday objects of our lives and connected via a continuous network". After that, Rogers [25] proposed a human centric UbiComp based on human creativity in using the environment to enhance their lives. This approach provides a solution of a specific UbiComp domain. In ref. [26], Caceres and Frida discussed the elements that make up UbiComp and the characteristics of the system to address the changing world. They point out two critical technologies for the growth of UbiComp which are Infrastructure-Cloud Computing and Internet of Things (IoT).

2.3 IoT Definition

The current internet has evolved into a network of interconnected objects so that they do not sense information and interact with the physical world. Rather, this development has expanded to provide services for information transfer, analysis and communication between them. This technological development, called the "Internet of Things", was appeared first in 1999 through Kevin Ashton in the supply chain management context [27]. Nevertheless, in the past decade, the term has become more widely inclusive, covering a wide range of applications such as health care, utilities and transportation [28].

IoT is the beginning of a new area of computing technology. It relies on the global incurable network in which different smart things communicate between them, with machines, and with environments. The IoT definition varies, in literature from an author to an other.

According to Van Kranenburg et al. [29], IoT is defined as a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual 'Things' have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network. According to Atzori et al. [30], the internet of things is based on three paradigms: internet-oriented (middleware), object-oriented (sensors) and semantic-oriented (knowledge). While this is a necessary distinction, because of the cross-disciplinary nature of the topic, the potential utility of the IoT can only be released in an application domain where the three paradigms intersect. According to cluster of European research projects on IoT [28], "Things" are active participants in business, information and social processes where they are enabled to interact and communicate among themselves and with the environment by exchanging data and information sensed about the environment, while reacting autonomously to the real/physical world events and influencing it by running processes that trigger actions and create services with or without direct human intervention. The Radio Frequency IDentification (RFID) group describes IoT as The worldwide network of interconnected objects uniquely addressable based on standard communication protocols [23]. According to Gubbi et al. [23], IoT is an interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications. This is achieved by seamless ubiquitous sensing, data analytics and information representation with cloud computing as the unifying framework. The definition provided by the ITU (International Telecommunication Union) is that IoT is a global infrastructure for the information society enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies [20]. According to Sharma et al. [21], the term Internet of Things (IoT) is a general concept for the capacity of the networked devices to sensor and capture data from all over the world and then distribute that data across the global internet where they may be analyzed and used for different useful applications. The IoT is smart machines communicating and interacting with other machines, objects, environments and infrastructures. According to Hukeriet al. [31], IoT is the growing network of objects or "things" integrated using electronics, sensors, software and connections to achieve higher value and service by communicating and service with the manufacturer operator or other interconnected devices. Each thing is distinctively by its embedded computer system but it is able to interoperate within the current internet infrastructure.

From all these definitions, we can conclude that IoT is a dynamic global network infrastructure, where physical and virtual "things" have identities, physical attributes, virtual personalities and use intelligent interfaces. These things are able to interact and communicate with themselves and the environment by exchanging data and information.

2.4 IoT Functional Blocks

A IoT system is composed of a number of functional blocks to facilitate various utilities to the system. These blocks are device, communication, service, management, security and application [32].

- The devices block provides monitoring , detecting, actuating, and surveillance activities. Devices exchange data with other connected devices and applications, or collect data from other devices. They also process data locally or send data to centralized servers or cloud-based application back-ends to process data or perform some tasks locally and other tasks within the IoT infrastructure depending on temporal and spatial constraints. IoT devices may also be of various types, for example, wearable sensors, smart watches, LED lights, automobiles, and industrial machines [32].

- The communication block ensures communication across devices and with distributed servers. IoT communication protocols generally function in the data link layer, network layer, transport layer and application layer [32].
- The services block in a IoT system supports different types of functions including device control, data management, device modeling, data delivery and device recovery services [32].
- The management block delivers various functions to govern an IoT system in order to research the IoT system's underlying governance [32].
- The security of the IoT system providing functions such as authentication, permission, confidentiality, message integrity and data security [32].
- The application block is the most critical in terms of users as it works as an interface that delivers the necessary modules to control and supervise various aspects of the IoT system [32].

2.5 IoT Architecture

The continuous evolution of IoT due to the association of devices with other areas such as cloud computing allowed the improvement of the sensors, actuators and the creation of smaller devices with a network connection [33]. In this context billions or trillions of heterogeneous devices connected are increasing. To manage data collected from these devices, a flexible layered architecture seems to be a better way. The basic model is the three layer architecture [34], [35], [36]. It consists of

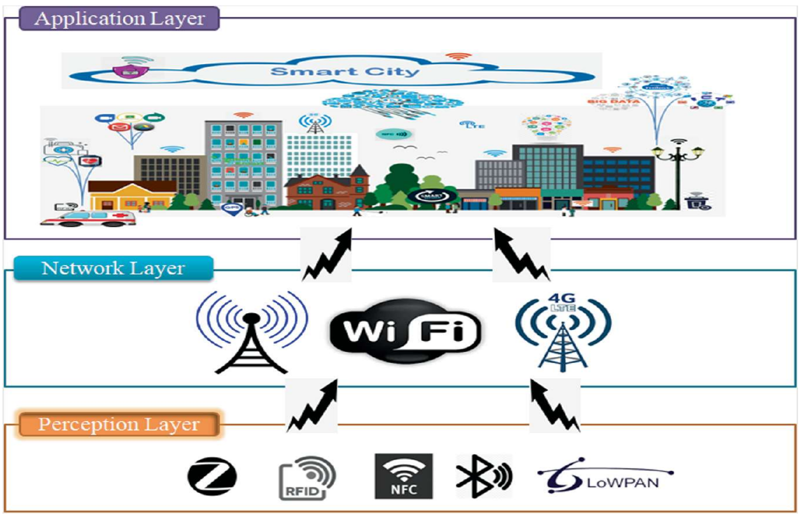


Figure 2.1: IoT architecture with three layers [38].

the perception layer, the network layer and the application layer (Figure 2.1).

The perception layer is the deepest layer of the IoT architecture. Similar to its name indicates, its objective is to collect data from the environment. All the data collection and detection part is done on this layer [37].

The network layer contains the data received by the perception layer. It collects the data from the inferior layer and sends it to the Internet. The network layer can only contain a gateway, with one interface connected to the sensor network and one connected to the Internet.

The application layer obtains information from the network layer and manages the application on a global base according to the information processed by the network layer. Based on the type of devices and their purpose in the perception layer and the way they have been processed by the network layer the application layer presents the data in the form of: smart city, smart home, smart transportation, vehicle tracking, smart agriculture, smart health and many other types of applications [34].

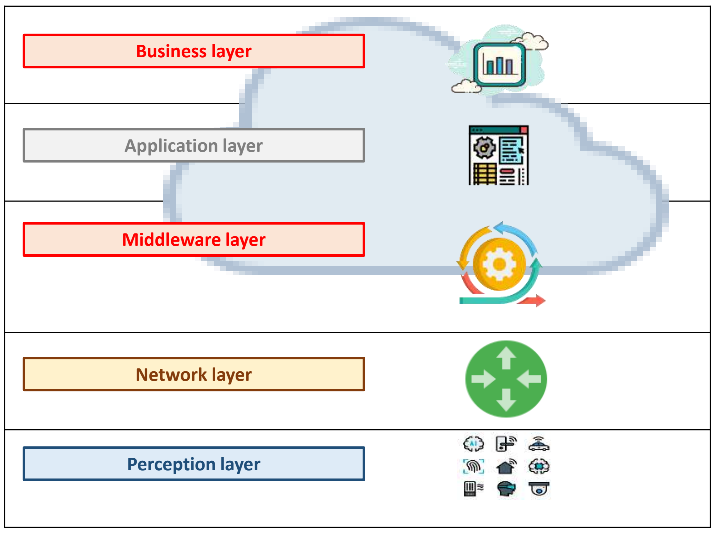


Figure 2.2: IoT architecture with five layers.

In the literature, some other models have been proposed that add more abstraction to the IoT architecture [34], [35], [30] where it extended the three-layer architecture to a five-layer architecture. They add two more layers, middleware layer and business layer (Figure 2.2).

The middleware layer links a service to its applicant according to addresses and names. It also links to the database to store, delivers the required services over the network wire protocols, received data and makes decisions from the network layer [34], [36].

The business layer supports the full range of operations and services of the IoT system. The business layer supports decision-making based on big data analysis [33]. In turn, the supervision and management of the four supporting layers is done at this layer. In this layer, the results of each layer are compared with the results of the other layers to improve services and preserve user privacy [34], [36].

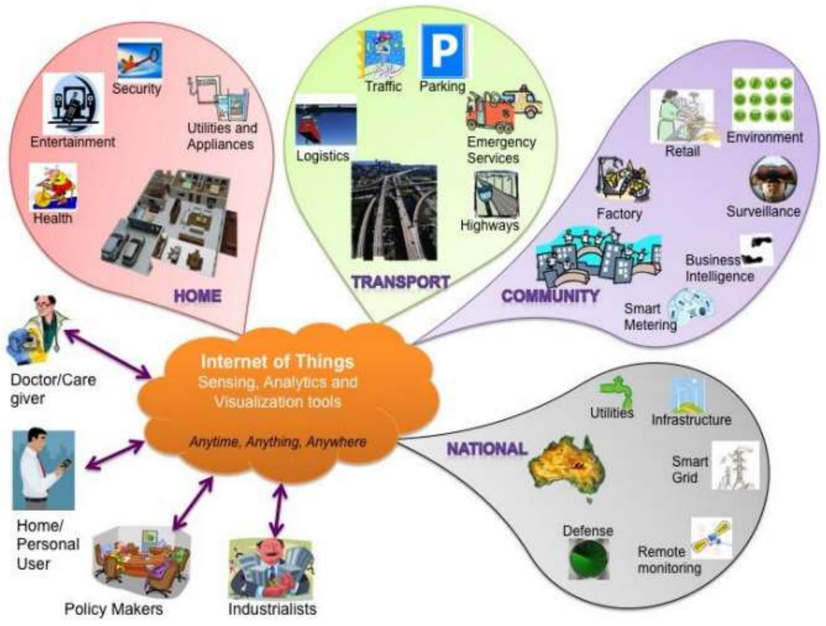


Figure 2.3: IoT applications domain [23].

2.6 IoT Applications

IoT plays a major role in enhancing the quality of our lives in several applications include transportation, healthcare, industrial automations etc. These applications classified according to Gubbi et al. [23] into four areas: personal and home, enterprise, public Services and mobile. (Figure 2.3).

2.6.1 Personal and Home

The information captured by the sensors is used by individuals only from their own network mobile [23]. WiFi is generally used as a network backbone that allows for a higher bandwidth data transfer (video) and higher sampling rates (sound). Among home applications, we mention :

- Control of domestic equipment including [21]:

  - Energy and water consumption: monitoring energy and water consumption to get cost reduction advice and resources.
  - Remote controlled appliances: Turn on and off appliances remotely to prevent accidents and save energy.
  - Intrusion Detection Systems: window and door detection and door openings and violations to prevent intruders.
  - Preservation of art and property: Surveillance of conditions inside the museums inside museums and warehouses.

- Healthcare:
  - Integrating sensors and actuators into patients and their medications for surveillance and follow up applications in hospitals.
  - Home surveillance systems for aged care, which allows to the doctor to monitor patients in their homes.

2.6.2 Enterprise

In a working environment as an enterprise, the data collected from the networks are only used by the owners and the data could be selectively liberated [23]. These networks are intelligent environments and we can cite smart home, smart city, smart agriculture, smart water and smart transportation [39].

2.6.3 Public Service

The information from IoT networks generally used to improve services, such as improving energy consumption in smart homes by continuously monitoring every power point inside the home and use that information to improve the manner in which electricity is consumed. In the video based internet of things, improve video monitoring where network camera surveillance applications help monitor targets and identify suspicious activities. In smart agriculture, improving the agricultural product where controlling the watering of agricultural land.

2.6.4 Mobile

There are two different domains of mobile applications: intelligent transportation and intelligent logistics. They are positioned in a distinct domain by the nature of the data exchange and backbone application needed [23]. Intelligent transportation [23] will allow large-scale Wireless Sensor Network (WSN) to be applied for online tracking of travel times, source-destination routing information, queue duration , pollutant and noise generation. Intelligent Logistics [40] includes the tracking of transported elements as the efficient planning of transports. The tracking of transported elements is carried out more locally, whereas the planning of transport is done by means of a large-scale IoT network.

2.7 IoT Challenges

2.7.1 Secure and privacy

Security is a main challenge since it covers very large scale networks which can see several types of attacks. The three physical components of the IoT: RFID, WSN and cloud are vulnerable to these attacks. Security is essential for any network [41], [42]. According to Juels et al. [43], the most vulnerability component is the Radio Frequency IDentification (RFID) as it allows the people and objects to be tracked and no high intelligence may be enabled in these devices.

2.7.2 Availability

The availability of the IoT needs to be realized in hardware and software stages to provide services at any time and in any place to clients. Software availability relates specifically to the capability of the IoT applications to deliver services to anyone at different locations simultaneously and in the hardware availability relates to the continuous existence of devices that support IoT functionality and protocols [33]. A various devices together with a different variety of communication protocols via TCP/IP or advanced software stacks could certainly manage the web services that will be displayed by different middleware solutions [44].

2.7.3 Reliability

Reliability is focused on improving the service delivery of IoT. However, IoT deployment is very complicated and consists of heterogeneous networks and smart devices, which leads to a reliability challenge. Reliability is needed to be implemented in software and hardware in each IoT layer. For an efficient IoT, the underlying communication should be robust as e.g. by unreliable perception, data collection, processing and transmission can lead to long delays, loss of data and possibly bad decisions, which can lead to disastrous scenarios and therefore can make the IoT seem less reliable [45].

2.7.4 Mobility

The IoT services delivered to mobile users is causing major challenges. These challenges include ensuring service continuity while users are on the move, service interruptions for mobile devices and the huge number of smart devices in IoT systems also require effective mechanisms for mobility management.

2.7.5 Performance

The IoT comprises an enormous number of components which provide services. The performance of IoT services is also affected by the performance of their components. These components require continuous supervision to ensure client demands are satisfied. Several measurements can be used to evaluate the performance of the IoT, such as processing speed, communication speed, device form factor and cost [33].

2.7.6 Management

Managing IoT resources includes, configuration, accounting, performance and security is a challenge because trillions of smart devices are connected. With the growing number of these resources, the development of light weight new management protocols to the standard management, that arise from the deployment of IoT in the coming years, become much more important.

2.7.7 Scalability

The addition of new functions and services for new equipments is a complex process within the IoT as various hardware platforms and communication protocols are available. In addition this scalability is achieved while not touching the quality of the current services.

2.7.8 Interoperability

The existence of heterogeneous and very complex network platforms in addition to the complexity between different devices types and their different communication technologies makes from interoperability a challenge. Interoperability must be addressed by application developers and IoT device producers to guarantee the service to all the clients, whatever the hardware platform they are using. Also interoperability must be addressed in the conception and construction of IoT services to satisfy clients needs [46]. Furthermore, interoperability must be addressed in the communication protocols.

2.7.9 Huge heterogeneous data

The wide range of devices connected to the IoT generates data of various types, sizes and formations. The variation and huge volume of this heterogeneous data create a serious challenge in the IoT.

2.8 Cloud Computing

2.8.1 Cloud computing definition

Several industry giants, standardization organisations and researchers have tried to define cloud computing in their understandings and opinions. Cloud computing is defined by the U.S. National Institute of Standards and Technology (NIST) as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [47]. This cloud model is composed of five essential characteristics (on-demand self-service, broad network access,resource pooling, rapid elasticity and Measured delivery), three service models (software as a service, platform as a service, and infrastructure as a service) and four deployment models (public, private, community, and hybrid) (Figure 2.4).

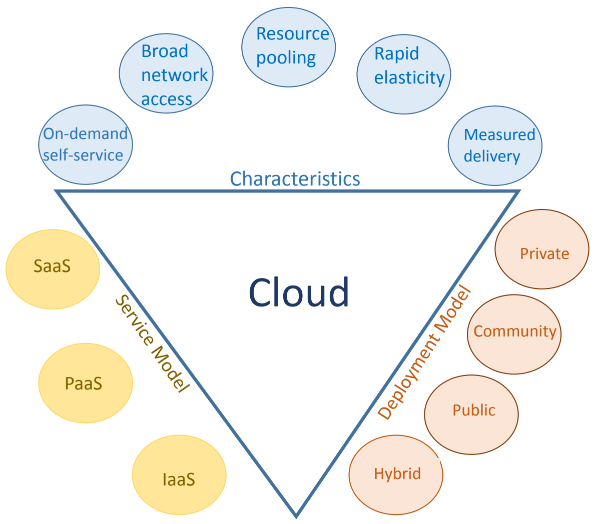


Figure 2.4: Scheme of NIST Cloud computing definition.

Cloud computing characteristics [47]

1. On-demand self-service: A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
2. Broad network access: These computing capabilities are distributed over the network (e.g. Internet) and used by different client applications using heterogeneous platforms (such as mobile phones cell phones, laptops and PDAs) located at a consumer site.
3. Resource pooling: A cloud service provider's computing resources are 'pooled' to serve multiple consumers using either the multi-tenancy or virtualization model, "with several physical and virtual resources dynamically allocated and reallocated based on consumer request".
4. Rapid elasticity: Capacity may be provisioned and released elastically, in

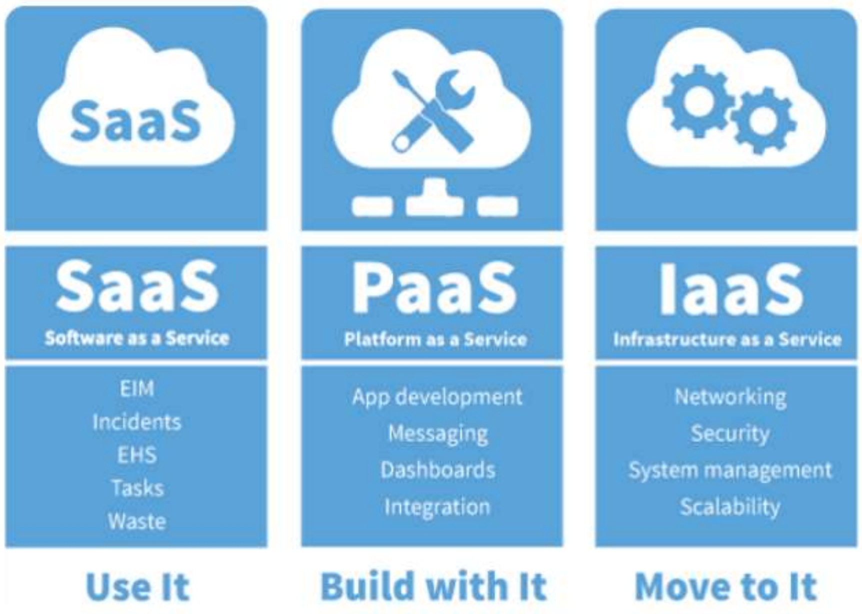


Figure 2.5: Cloud computing service models [48].

some with automatic release, to move quickly outward and inward as demand dictates. To the consumer, the capacity available for delivery often seems unlimited and can be appropriated in any quantity at any time.

5. Measured delivery: Cloud systems automatically check and optimize resource allocation by leveraging a measurement capability at some level of abstraction that' s appropriate for the type of service (e.g, storage, processing, bandwidth and active user accounts). Resource usage may be monitored, controlled and reported, ensuring transparency for both the provider and consumer of consumer of the service used.

Cloud computing service models [47]

1. Software as a Service (SaaS): The ability for cloud consumers to use their ap-

plications on a hosting environment, which can accessible via networks from from various clients (e.g, browser, PDA, etc.) (Figure 2.5). Cloud consumers do not control the cloud infrastructure that often uses a multi-tenant system architecture, i.e. different applications of cloud consumers are tructured in a single logical environment on the SaaS cloud benefit from economies of scale and optimization in terms of speed, security, availability disaster recovery and maintenance.The user is not required any storage, installation and maintenance of the application. However, Internet connectivity is needed to access the service that is rented by the SaaS service on the cloud. Some examples of thees services SalesForce.com, Google Mail, Google Docs, etc.

2. Platform as a Service (PaaS): The ability offered to the consumer to provide a platform for the development of cloud services and applications (Figure 2.5). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

3. Infrastructure as a Service (IaaS): The ability offered to the consumer is to provide processing, storage, networks and other computing resources in which the consumer is in a position to implement and run arbitrary software,including operating systems and applications (Figure 2.5).

A fourth service model, called Networks as a Service (NaaS), was added by Aazam et al. [49]. The Networks as a Service (NaaS) represents the ability to provide one or more virtual networks to users. The user could have as many network number as needed, with the appropriate segmentation and policy enforcement. With NaaS, the user may also have heterogeneous networks.

Cloud computing deployment Models [47]

1. Private cloud: The cloud infrastructure is operated only a single organization with multiple consumers. It can be managed by the organization or a third party, whether located on-site or off-site premise. There are several aspects to the reason for introducing a private cloud in an organization. Security problems, including data privacy and confidentiality, the cost of moving data from one infrastructure to another, optimise the use of existing internal resources and organizations always demand complete control over critical activities that reside behind the cloud.

2. Community cloud: The cloud infrastructure constructed by multiple organizations, which share the same cloud infrastructure as well as policies, requirements, values and concerns. The cloud infrastructure could be either owned by a third party provider or within one of the organizations of the community.

3. Public cloud: The cloud infrastructure composed of two or more different cloud infrastructures. Cloud infrastructure that' s provided for public use. The public cloud is used by public consumer cloud and the cloud service supplier has complete ownership of the public cloud along with the provider's policy, costing, profit and billing model. Most popular from cloud services public clouds, such as Amazon EC2, S3,Google and AppEngine.

4. Hybrid cloud: The cloud infrastructure composed of two or more different cloud infrastructures. Organizations employ the hybrid cloud model to optimize their resources in order to enhance their core competency and to control their core business on-premises via the cloud. The hybrid cloud has raised the issues of standardization and interoperability of the cloud.

2.8.2 Cloud computing architecture

The cloud architecture is composed of three layers: infrastructure, platform and application [50](Figure 2.6).

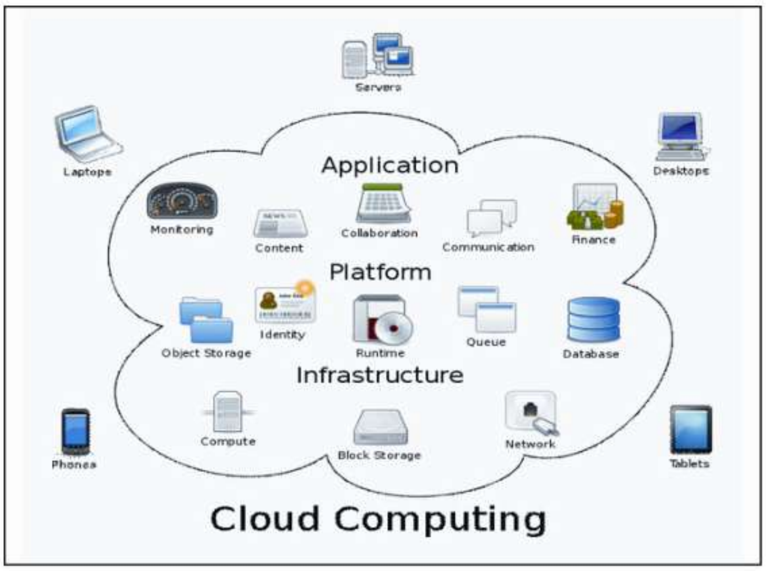


Figure 2.6: Cloud computing architecture [50].

- Infrastructure layer, is the most basic layer. This layer delivers the processing, storage, networking and other computing resources. Cloud service customers can deposit and execute operating systems and software for their software to their infrastructure.
- Platform layer, delivers superior abstractions and services for applications in the same integrated development environment. This layer includes an execution environment and middleware to support the deployment of applications using programming languages and tools cloud service.

- Application layer, is the upper layer. The application layer can sense environment data and send requests to the cloud simultaneously to process and obtain sensor information results [49]. It is also necessary to re-post information to the IoT, data obtained from the sensor layer and data analysis for additional processing [51], [52].

2.8.3 Cloud computing challenges

The integration of internet of things (IoT) and the cloud computing make possible the storage, the process and the analysis of the massive IoT data generated by the different devices. However, there are challenges that need to be addressed to allow the cloud to prevail for the good of the world in general and humanity in particular. These challenges will be presented in what follows.

Security The variety of applications and the heterogeneity of devices in an IoT environment made it difficult to ensure the privacy and security of the data generated by these devices. To address security challenges in cloud computing, the following considerations are important [53]:

- End-user trust and privacy.
- Source authentication between nodes.
- Impenetrable communications between sensors, compute and brokerage nodes.
- Identification and protection of systems from malicious attacks.
- Robust data management and tamper resistant databases .

Current research addresses issues such as malicious detection and recovery, identification and protection against attacks, prevention of malicious threats, protection of user information against theft and dynamic mutual authentication [54], [55].The existing research are operated at a limited angle and the computational capabilities of the two edges and distance resources have not been fully exploited [56].

Protocol support In order to get different things connected to the Internet, various protocols are going to be used. So, although they might be similar entities and operate on different protocols. The solution to this problem can be the standardization of protocols.

Energy The expansion of data collection and processing resulted in an energy consumption growth in the cloud data centers of 20 to 25% each year [57]. To solve this problem, they directed to the distributed cloud, which conducted to the increase in importance of fog and edge computing platforms. As the massively expanding number of IoT devices [58], the communication of all devices with the cloud result in a higher power consumption. Furthermore, smaller IoT devices with low computing power, storage and battery are being developed [59]. For example, the change of batteries from time to time in order to power the cameras. Likewise the encoding of the videos is more complex than decoding. The point is that for an efficient video compression, the encoder has to analyze the redundancy in the video [60].

Reliability IoT devices are relying on the cloud to operate providers for time-critical applications and the impact would directly reflect the program's output [61].

Resource allocation Resource allocation in distributed systems is a difficult challenge in the scale of the current data center. The varying character of network devices, devices components and communication technologies in large scale distributed systems results in the complexity of resource management techniques growing [62]. In the other hand, due to the variety of devices that this leads to the production of different types of data in addition to the amount that will be produced, it is difficult to predict the resources they will need in the cloud. Several flat forms developed to solve the problem of resource allocation. Such as Mesos determines the number of resources to allocate to each network according to the constraints, while the latter in turn decide which offers to be accepted. So there is a need for new approaches to resource allocation which help to ensure the stability and efficiency of these systems. Resource allocation is a critical concept in distributed systems, however it must ensure that these systems have high performance, latency sensitivity, reliability and energy efficiency [63], [64].

Quality of service Quality of service (QoS) is a critical challenge in cloud computing systems, as it can be predicted by the system performance during run time [65]. QoS settings that may be used to measure system performance such as execution time, cost, scalability, elasticity, latency and reliability, etc [63]. QoS is increasingly important when it takes into account cloud services, because damaging the QoS in one of them can dangerously affect the QoS of the complete computing system. The following are some of the research challenges cited in [66] that affect the realization of QoS efficiently.

1. The non availability of cloud resources to execute an application during run time, which increases the execution time and reduces the system performance.
2. Making effective resource management mechanisms that take into account SLAs (Service Level Agreements) reduces the rate of SLA violations and helps to improve the performance of the computing system.
3. The existence of varying SLA standards for the various cloud providers means

that and a centralized SLA standard is needed to attain the goal of a multi-cloud environment.

4. Find the trade-off between the various QoS needs due to the vast amount of IoT applications run on cloud systems using supervised/unsupervised learning techniques based on AI(Artificial Intelligence) or predictive models.

2.9 Fog Computing

2.9.1 Fog computing definition

The storing and processing of data from various IoT sensors is a critical challenge in an IoT system. Traditional cloud based IoT systems are posed with challenges due to the large scale, heterogeneity and high latency observed in some cloud systems [67]. Consequently, a novel computing paradigm, namely "fog computing", has been introduced as a complement to the cloud solution. According to NIST [67], fog computing is a layered model for enabling ubiquitous access to a shared continuum of scalable computing resources. The model facilitates the deployment of distributed, latency-aware applications and services and consists of fog nodes (physical or virtual), residing between smart end-devices and centralized (cloud) services. The fog nodes are context aware and support common data management and communication system. They can be organized in clusters either vertically (to support isolation), horizontally (to support federation) or relative to fog nodes latency-distance to the smart end-devices. Data processing tasks that require real time processing of data from end devices can be performed by nearby fog nodes, leading to low transmission latency [68]. In addition, the fog computing is the form of distributed computing which functions as the middle layer between IoT devices and cloud data centers [69].

Characteristics of fog computing

1. Contextual location awareness and low latency

   Fog nodes are located in close proximity to IoT devices, implying low latency services and applications. Moreover considerably more rapid analysis and response to data generated by devices compared to a centralized cloud.

2. Geographical distribution

   The services and applications that the fog focuses on need widely distributed deployments to ensure QoS for mobile and immobile devices [67]. The fog network geographically distributes their nodes and sensors in the scenario of a different phase environment, for example, the healthcare monitoring system [70].

3. Very large number of nodes

   The wide geographical distribution, as reflected in sensor networks for the most part and the Smart Grid in particular [67].

4. Large scale sensor networks

   The fog is distributed resources and has a distributed storage that need environmental monitoring, in close smart grid applications.

5. Support for mobility Fog applications communicate directly with mobile devices, so they support mobility techniques. Like the LISP protocol that decouples host identity from location identity with a dispersed indexing system [67]

6. Real time interactions

   Fog applications necessitate real time interactions, such as real time transmission for traffic monitoring systems, control of a sensitive process on an oil platform with fog edge devices or sensors...etc.

7. Widespread of wireless access

   The wide range of wireless sensors distributed and connected in the network requires distributed analysis and processing. This is why the fog is very well suited for wireless IoT access networks.

8. Heterogeneity

   Fog nodes are very varied in their nature and will be employed in a large variety of environments which include a variety of devices and have different network communication capabilities.

9. Interoperability and federation

   Fog elements need to operate in an interoperable environment to ensure a wide range of services such as data streaming and real time processing . These services must be federated between domains.

10. Support for real time analytics and interplay with the cloud

    The fog nodes are located nearer to the source that generates the data. This location provides low latency and low context awareness, however the cloud provides global centralization. Analytics big IoT data requires the localization of fog for real-time stream analysis and the globalization of the cloud for historical lot analysis of big IoT data. Additionally, fog is well adapted to handle video streaming in small TV support devices, surveillance sensors, live game applications and other applications that required low latency services in near proximity [71].

11. Scalability and agility of federated, fog-node cluster

Fog Computing is adaptive in nature, at the cluster or cluster of clusters, with support for elastic computing, resource pooling, data load changes, and network state variations, to list some of the adaptive features supported [72].

2.9.2 Fog node

Fog computing employs, in addition to centralized cloud data centers, a vast number of smaller capacity resources closer to the edge of the network, termed fog nodes [72]. According to NIST [67], fog nodes are middleware elements within the smart terminal and access network. Fog nodes can be physical or virtual and are coupled to smart devices or access networks. Fog nodes typically deliver some form of data management and communication service between the edge layer in which the smart terminals reside and the cloud. Fog nodes, in particular virtual nodes, also called cloudlets, can be federated to provide a horizontal extension of capability over distributed geolocations.

Fog node attributes Several attributes are added to the fog node characteristics to support the deployment of fog computing capability which are [72]:

1. Autonomy, fog nodes are able to function autonomously, with decisions made locally, at the node or cluster level.
2. Heterogeneity, fog nodes are available in a variety of form factors and can be employed in a range of environments.
3. Hierarchical clustering, fog nodes are adopting on hierarchical structures, where different layers providing diverse subsets of service functions and working collaboratively like a continuum.
4. Manageability, fog nodes are managed and engineered by complex systems capable of executing most routine most routine functions automatically.

5. Programmability, fog nodes are programmably embedded at multiple levels, by multiple parties.

Service Models Same case as the cloud computing, these types of service models can be implemented in the fog nodes [72]:

1. Software as a Service (SaaS): Users are using the fog provider's applications, which they are running on a cluster of federated fog nodes managed by the provider. Intelligent objects can be accessed by the fog node's applications via a client or program interface. The infrastructure underlying the fog node, such as the network, servers, operating systems, and storage, is invisible to the user.
2. Platform as a Service (PaaS): The fog service provider uses programming languages, libraries, services and tools to provide services to clients. Witch they may use the platforms of federated fog nodes.
3. Infrastructure as a Service (IaaS): Users can run arbitrary software, which can include operating systems and applications that leverage the infrastructure of the fog nodes forming a federated cluster. Users do not monitor or control the underlying infrastructure of the fog node cluster, however, they do have control over operating systems, storage and deployed applications.

Deployment models Similar to cloud computing, the following deployment models can be applied to fog nodes computing [72]:

1. Private fog node

   A fog node is operated by a single organization with multiple consumers. This node can be independently controlled, managed and operated by the organization, a third party, or a hybrid of the two, and it can be located either inside or outside of the organization.

2. Community fog node

   A fog node that is provisioned by a consumer community of organizations that have common concerns. This node can be independently controlled, managed and operated by one or more of the organizations in the community, a third party or a combination of them, and it can be located either inside or outside of the organization.

3. Public fog node

   A fog node which is provisioned for open use by the public generally. This node can be independently controlled, managed and operated by a company, university, or government organization, or some combination of them, and it can be located either inside or outside of the organization.

4. Hybrid fog node

   A fog node which is a combination of two or more distinct fog nodes (private, community, or public) that remain unique entities, however, they are linked by a standardized or proprietary technology that makes data and applications portable.

2.9.3 Fog architecture

In general, most of the research projects carried out on fog computing have mostly represented as a three-layer model [53], [73], [74], [75]. Other teams proposed models with four layers [76], [77], five layers [78], six layers [49] and seven layers [79].

In addition, the OpenFog Consortium [80] has developed a detailed architecture reference of the N-layer, which is regarded as an improvement of the three-layer model. Another three-dimensional architecture (The device dimension, The system dimension and The functionality dimension) is proposed by [81] However, we will search for to a three-layer architecture in the following.

In the three-layer model, fog computing extends the cloud service to the edge of the network, in which a layer of fog is introduced between the terminals and the cloud. Figure 2.7 illustrates the hierarchical architecture of fog computing. The hierarchical architecture is comprised of the three following layers:

1. Terminal layer: It is the layer nearest to the terminal user and physical environment. It comprises a number of widely distribution IoT devices, such as smart vehicles, sensors, cell phones and smart cards. They are responsible for sensing and transmitting data to the higher layer for processing and storage.
2. Fog layer: It is situated at the edge of the network and largely distributed between terminals and the cloud . It is comprised of a variety of fog nodes, which typically consists of gateways, routers, access points, specific fog servers and switches,etc. This layer is important for the interaction and the collaboration with the cloud layer, which is connected with this layer.
3. Cloud layer: With multiple high performing servers and storage devices and delivers different application services. It has strong computational and storage abilities in order to take care of profound computational analysis and storage of a big IoT data.

2.9.4 Fog computing challenges

1. Security and privacy

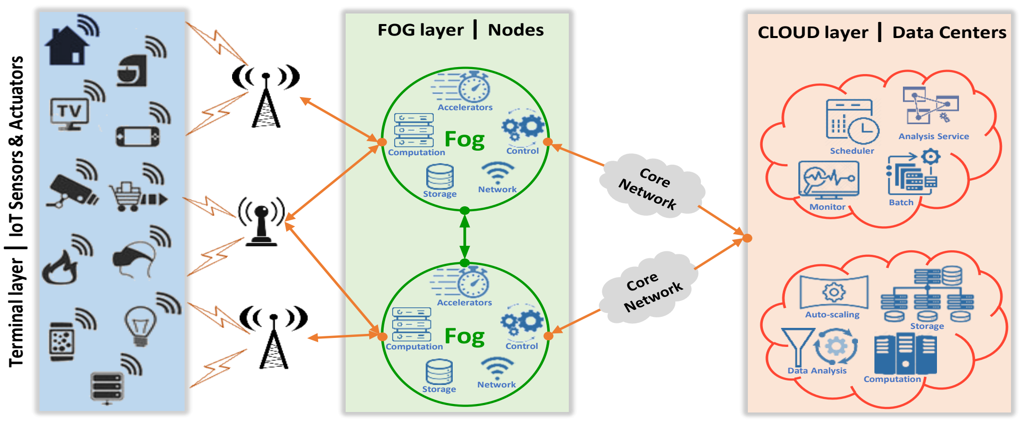


Figure 2.7: Architecture of fog computing.

The security of fog computing devices is challenging due to the fact that they are operated in non-strict locations. The protection and monitoring of these devices is vulnerable to attacks that could be used to compromise the fog device system in order to perform malicious tasks such as data hijacking and eavesdropping. The security solutions proposed for the cloud cannot support fog computing due to the fact that fog devices operate at the edge of networks. The operating environment of fog devices can address many threats that do not exist in cloud computing. The major attacks which can be launched against fog computing cited in [82] are man in the middle, authentication, distributed denial of service, access control and Fault tolerance.

2. Control and management resources

To ensure the QoS the fog computing should perform a provisioning to prevent the resources to be used in order to provide the service mobility. The major challenge is that the mobility of the end nodes, since these metrics such as bandwidth, storage, computation and latency will be modified dynamically [82]. In addition, resource management is a challenge due to the fact that the fog computing must manage the sharing and discovery of resources used by Cloud applications and manage the sharing of resources used by the devices in the terminal layer.

3. Programming platform

   In the fog computing, the computing is performed in the user end edges nodes that are usually probably run heterogeneous platforms and generally different from another, so programming in such heterogeneous platforms is a major challenge.

4. Energy management

   Fog computing systems comprise multiple distributed nodes, so energy consumption is expected to be higher than their cloud counterparts. Hence, much effort is required to develop and optimize new energy efficient protocols and architectures in the fog fog paradigm, e.g., efficient communication protocols, computing and network resource optimization [82].

5. Fog networking

   The fog network gets heterogeneous, situated at the edge of the network and with extensions to the cloud computing functionality. The fog networking requirement is to interconnect each required component to the node to maintain and insure the quality of service in the core network connectivity and service delivery on all these components. In the increasing use of IoT in wide scale use, this use might not be straightforward [70].

6. Quality of Service (QoS)

   The quality of service aspect is very important in fog computing and causes a challenge, according to Anawar et al.[70] classified these challenges into four dimensions, reliability, delay, connectivity and capacity, that are discussed as follows:

- Reliability, is most important for data transmission and security in the core network. In addition, it is required to periodic surveillance by means of checkpoints in order to recover from a fault.
- Delay, in fog computing is a difficult challenge. As the deployment of a fog network for applications, which are sensitive to latency, needs a real time streaming and processing response.
- Connectivity, for the fog networking environment requires providing partitioning and clustering capabilities for cost minimization, data reduction, and an extension of connectivity methods.
- Capacity, the capacity of QoS in [70] is categorized into two groups, the first group being network bandwidth and the second group being storage capacity. These are very important factors in enabling and maintaining effective bandwidth and storage operations. In addition, real-time response, fog node mobility, and large fog computational volume are all factors that must be considered to save maximum bandwidth with low latency.

2.10 Conclusion

In this chapter, IoT definition, its architecture, the main applications and its challenges are provided and discussed in addition to modern cloud and fog computing paradigms that have emerged to support the deployment of IoT based applications.

In front of the great interest of IoT, the presented challenges are to be solved especially the huge heterogeneous IoI data challenge. The process of this big data in metric space presents the disadvantage of data overlap as well as its process in the multidimensional space. A pre-processing step, in order to diminish data overlap during big IoT data store, will be of a great usefulness. The analytics methods are supposed be a good candidate for pre-processing big IoT data. Among these analytics methods, clustering methods which group objects into homogeneous sets or clusters.

3 Clustering Methods of Big IoT Data

3.1 Introduction

The development of big data and IoT is accelerating rapidly and affecting all areas of technology and businesses. The growth of data produced via the IoT has played a major role on big data. The widespread use of IoT big data has made it difficult to analyze. This necessitates development of IoT specific data analytics solutions which can handle the heterogeneity, dynamicity and velocity of the IoT data [10].

Data mining is the process of extracting useful information or to find out hidden relationship among data. This information or knowledge is very helpful for business organizations to grow their business as it is helpful in decision making. Data mining technology has come across several stages [83].

Big data analytics enables data miners and scientists to analyze huge amounts of unstructured data that can be harnessed using traditional tools [84]. These tools are developed using data mining algorithms, based on a specific scenario, such as the prediction method, association rule methods, classification methods and clustering methods [85]. Clustering methods are an essential branch of the data mining family that has been largely applied in IoT applications such as outlier detection, finding similar sensing patterns, and segmenting large behavioral groups in real time [86].

In this chapter, we present definitions of big data and big IoT data. After that, We focus on different clustering methods since one of these methods (Density-Based Spatial Clustering of Applications with Noise algorithm) is used, in this work, in the pre-processing step which will allow a parallel processing of big IoT data. In the end of this chapter, other big IoT data analytics methods will be provided after a comparison between the different clustering methods.

3.2 Big Data Definition

The usage of the term "big data" officially arrived in the computing field in 2005 by RogerMagoulus of O'Reilly to depict the massive volumes of data that cannot be managed and processed by traditional data management techniques as it gets too complex and vast in size [87], [88]. The use of the term "big data" has occurred in previous literature, although it is a comparatively new one in business and IT (Information Technology) [89]. Several studies related to big data are available. One of these studies, "Digital Universe" [90] defines big data technologies as a new generation of technologies and architectures that aim to exploit a massive volume of data with different formats by enabling high-speed capture, discovery and analysis.

Other studies describe big data in three dimensions 3Vs "volume, variety, velocity" [91] are regarded as the essential concentrated when defining big data and it is in consistent with Madden's [92] definition of big data which say: this is data that is too big (volume) from different sources, too fast (velocity) as it must be processed rapidly, and too hard (variety) to process by existing tools.

Another approach to defining big data in expanded on the 3 Vs to 4 Vs where the aspect of "veracity" for data that is too uncertain [93].

Attribute, "value", is introduced to give it a greater meaning the 5 Vs of big data. Importantly, big data is critical to organizations as it facilitates the collection, storage management and manipulation of massive amounts of data in order to data to make useful decisions. This qualifies the inclusion of "value" as the fifth attribute of big value of data collection applies to the intended process or the predictive analysis. The value of data is closely associated with other attributes of big data such as volume and variety [94].

In the preceding definitions, the attributes of big data stay the same in an enterprise network outside of the "variability" that takes the network infrastructure into account especially on the integration, evolution of the data and the model (variability, which takes care of varying data and associated models.) [95].

The appearance of technologies that enable real-time communication with objects that move in real time, known as spatial and temporal database models that interact, with data representation calls for a more refine approach to description. The latter integrates the seventh attribute, which is a "visualization" [96]. This attribute ensures the readability and accessibility of data presentations that need many spatial and temporal parameters and relationships associated with each other [97].

Another study added the 8 Vs of big data introduce a " Validity". It may mean that the data should be clean, accurate,precise , specific, reliable, valid and useful for future processing. Every organization should validate the data if it needs to take the right decisions for the future based on the data collected by the devices. So, validity is regarded as an important factor for big data [98].

In the case of the 9 Vs of big data, "Vulnerability", the data violation is a critical concern in today's age of technology. Hackers are continually and constantly hacking into systems and databases to gain access to information. The big data

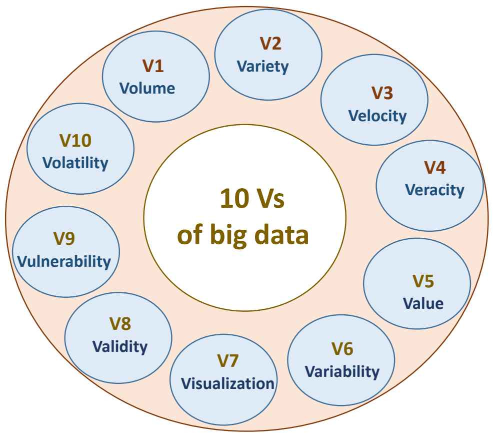


Figure 3.1: Brief description of the 10 Vs of big data.

violation is a big breach and hence, vulnerability is also a challenging and critical characteristic of big data, for securing information against unauthorized persons and unauthenticated access is a basic need.

Further study of big data added the 10 Vs of big data, "volatility" [99]. They consider the introduction of "volatility" that is affected by the lifetime of the data to answer the following question: How long will the data be regarded as valid and how long long it needs to be stored?. The brief description of the 10 Vs of big data is shown in figure 3.1. For the case in [100], the researchers presented the big data intelligence, which refers to the ensemble of concepts technologies, tools and systems which are able to approximate human intelligence in the management and processing of big data.

3.3 Big IoT Data Definition

Big IoT data, term appeared as a result of several searches and technological development. Recently, Sun et al. [100] added an important feature forming the 10 Vs. It captures all technologies, systems, platforms and facilities which support the big data processes. In this context the 10 Vs of big data capable of integer IoT data, where in the area of IoT, the continuous increase in the number of IoT devices has led to the production of huge amounts of data. According to statistics [101], the number of devices will be increased by 1 trillion by 2030. As these devices are numerous, they became a source of big data called "big IoT data". A most notable characteristic of IoT is its analysis of data on "connected objects" [85]. The analysis of big IoT data needs various methods for the processing and the storage of a large amount of IoT data.

3.4 Clustering Methods

Clustering, is a data mining technique that is used as a major method of data analysis. Clustering employs an unsupervised learning approach and generates groups for given objects based on their distinctive significant features [102]. Clustering is the process of grouping a set of physical or abstract objects into classes of similar objects. A cluster is collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. Clustering algorithms can be categorized into partitioning-based algorithms hierarchical-based algorithms, grid-based algorithms, model-based algorithms and density-based algorithms [103].

3.4.1 Partitioning clustering algorithm

Splits the data points within k partitions. Each partition is considered as a cluster. Partitioning is performed based on some objective functions. One of these function minimizes the square error criteria which is calculated as:

$$E = \sum \sum ||p - m_i||^2 \tag{3.1}$$

where p the point of a cluster and m_i the mean of the cluster. Among these partitioning methods, we can cite k-means [104] and FCM (Fuzzy CMeanS) [105].

k-means [10] Divides a given data set into k different clusters. This is done by first, choosing k random points in the dataset as the initial clusters centroids, then, allocate each data point to the appropriate of these clusters by adjusting the center. The process is repeated with the output as new input arguments until the centroids converge to stabilized points.As the final clustering results are highly dependent on the suitable centroids, the whole process is performed several times with different suitable initial parameters. For a fixed dataset size it might not be a problem, however in the context of IoT data the characteristic of the algorithm causes significant computational overhead. k-means convergence for clustering with randomness not only this process need time, it also means that k-means may produce lower quality.

Fuzzy C MeanS (FCM) [105] FCM is based on the k-means concept to partition the data set into clusters. The procedure is as described below:
Compute the cluster centroids and the objective value and initialize the fuzzy matrix. Compute the membership values stored in the matrix. If the objective value between consecutive iterations is less than the stopping condition, stop. This process is continuous until a partition matrix and clusters are formed.

3.4.2 Hierarchical clustering

A technique of clustering which divides the similar dataset by building a hierarchy of clusters. This method is based on the connectivity approach of clustering algorithms. It is based on the distance matrix criteria to group the data. It constructs clusters step by step. In hierarchical clustering, there are two approaches: Clustering agglomeration (top-bottom) and Division (bottom-up). In agglomerative approach, suitable object is selected and successively merges neighboring objects according to the distance to the minimum, maximum and average. The process is continuous until a desired group is formed. The division approach deals with set of objects as a single cluster and divides the cluster into other clusters until the desired number of clusters is formed [103]. Among these algorithms we can cite: Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) [106], Clustering Using REpresentatives (CURE) [107] and Robust Clustering algorithm for Categorical attributes (ROCK) [108].

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) [106] It is an agglomerative hierarchical clustering algorithm especially adapted for very large large databases [109]. The BIRCH process starts by building CF-tree. Condense the data by rebuilding the CF-tree with a larger tree. Then one of the existing clustering algorithms is used on the CF-tree leaves. After additional passages performed on the data set and reassign the data points to the centroids closest to step above. This process continues until k cluster steps are formed.

Clustering Using REpresentatives (CURE)[107] It is an agglomerative hierarchical clustering method that creates a balance between centroid and all point approaches.[109]. A Divisive approach hierarchy is used and it selects well dispersed points of the cluster and then shrinks to the cluster center by a specified function. Adjacent clusters are consecutively merged until the number of clusters reduced to the desired number of clusters. The procedure is given by: Initially every point is in separate clusters, each cluster is defined by the point in it. The representative points of a cluster are generated by first selecting well dispersed objects for the cluster and then shrinking or shifting to the cluster by a specified factor. At each step of the procedure, two clusters with the closest pair of representative points are selected and merged together to form a cluster.

Robust Clustering algorithm for Categorical attributes (ROCK) [108]
It is an agglomerative hierarchical clustering algorithm based on the notion of links [109]. It is a hierarchical clustering algorithm where forming clusters, it uses a link strategy. Links from bottom to top merge to form a cluster. The procedure is given by: First considers a set of points in that each point is a cluster and calculate the links between each pair of points. Build a heap and maintain the heap for each cluster. A quality measure based on the criterion function be computed between the pairs of clusters. Merge the clusters that have a maximum value of criterion function.

3.4.3 Grid-based algorithms

The grid based algorithm is based on partitioning the dataset into number of cells to form a grid structure. The clusters are formed based on the grid structure. To build clusters, the grid algorithm uses subspace and hierarchical clustering techniques [103]. Among these algorithms we can cite: STatisitcal Information Grid based method (STING) [110], CLustering InQUEst (CLIQUE) [111] and Merging of Adaptive Intervals Approach to SpatialData Mining (MAFIA)) [112].

STatisitcal Information Grid based method (STING) [110] It is similar to the BIRCH hierarchical algorithm [106] for building a cluster with spatial databases. The process starts by stored the spatial data in rectangular cells using a hierarchical grid structure. Then each cell is partitioned into four child cells at the next level with each child corresponding to a quadrant of the parent cell. The probability is computed of each cell being relevant or not. If the cell is relevant, apply the same calculations on each cell one by one. Finlay, find the regions of the relevant cells to form a cluster.

CLustering InQUEst (CLIQUE) [111] A subspace clustering algorithm of numerical attributes where the bottom-up approach is employed to build clusters. The algorithm is described in this way: Consider a set of data points,in a one pass, apply width to the set of points to form the grid cells. Rectangular cells in a subspace whose density exceed τ are placed in equal grids. The process is continued recursively to form $(q-1)$ dimensional units into q dimensional units. The subspaces are connected to each other to form cluster of equal width.

Merging of Adaptive Intervals Approach to SpatialData Mining (MAFIA)) [112] It is a variant of the CLIQUE algorithm [111]. Unlike the CLIQUE algorithm, it uses a fixed cell size grid structure with an equal number of cells. Using a grid structure of fixed size cells with an equal number of cells in each dimension of bins in each dimension, it constructs an adaptive grid to improve the quality of the clustering. The algorithm is described in this way: In a single pass, an adaptive grid structure was built by considering a set of all points. Calculate the histogram by reading blocks of data in memory using bins. The bins are grouped based on dominance factor α. Choose the bins that are r α times denser than the mean as p candidate dense units (CDUs). Recursively, the process continues to form new p-CDUs and merge adjacent CDUs into clusters.

3.4.4 Model-based algorithms

In a data collection, data points are connected with each other according to different strategies such as statistical methods, conceptual methods and robust clustering methods. Two approaches to model-based algorithms are available: the neural and the statistical approach [103]. Among these approaches, we present Self Organized Map algorithm (SOM) [113] as neural approach and Model based clustering algorithm (COBWEB) [114] as statistical approach.

Self Organized Map algorithm (SOM) [113] Neural networks consider each cluster as a neuron, and the input data are also considered as neurons. Each neuron connection is assigned by some weight, that is randomly initialized before learning these weights in an adaptive manner [115]. SOM [113] is one of the most widely used algorithms. The SOM is considered as a two layer. Each neuron represented by n-dimensional weight vector, $m = (m_1, \ldots, m_n)$, where n is equal to the dimension of input vectors. The neurons of the SOM are itself cluster centers, hoverer to accommodate interpretation the map units can be combined to form bigger clusters. The SOM is trained iteratively. In each training step, one sample vector x from the input data set is chosen randomly. The distance between it and all the weight vectors of the SOM is calculated using a distance measure. After finding the best matching unit, the weight vectors of the SOM are updated so that the best matching unit is moved closer to the input vector in the input space [116].

Model based clustering algorithm (COBWEB) [114] The COBWEB algorithm gives a clustering dendrogram called classification tree which characterises each cluster by a probability description [117]. It is known as an incremental learner since when a data object is entered, the nodes of the tree are restructured.

In some situations, it can change the entire structure of the tree considerably. COBWEB uses the category utility (CU) measure [118] as the criterion function for determining partitions in the hierarchy. The expected value of CU used in COBWEB is defined as $P(A_i = V_{ij} \setminus C_k)^2$. If the the given data object is in a cluster C_k, the CU value implies the probability that A_i as the value V_{ij} which signifies the probabilistic match of the data object to the cluster. The role of category utility is to make a trade-off between maximizing intra-class similarity and inter-class dissimilarity. CU was used as the basis for incremental clustering [118].

3.4.5 Density-based algorithms

Density-based clustering algorithms try to find clusters based on density of data points in a region [116]. The main idea of density based clustering is that for each instance of a cluster the neighborhood of a given radius *eps* has to contain at least a minimum number of instances $MinPts$ [116]. The data objects are categorized into: core points, border points and noise points. All core points are interconnected based on the densities to form a cluster. They can find the cluster based on the regions that are growing at high density. These are the one-scan algorithms. It is capable of getting the arbitrary shaped clusters and handle the noise. One of the most well known density based clustering algorithms is the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [119] and the Ordering Points To Identify the Clustering Structure(OPTICS).

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [119] It is a connectivity based algorithm which consists of three points namely core, border and noise (Figure 3.2). Given a distance threshold r in DBSCAN the distance threshold is named *eps* and a density threshold k in DBSCAN the density threshold is named MinPts .

1. The density of a point x_i is defined as the number of points k_i that are within a radius r around x_i.
2. If $k_i > k$, the corresponding point x_i is considered a core point.
3. Two points are considered directly connected if they have a distance of less than r.
4. Two points are density connected if they are connected to core points and these core points are in turn density connected.
5. A border point has less MinPts in *eps*, this point is is in the neighborhood of a core point.
6. A noise point is defined as any point that is not a core point nor a border point.

These definitions allow to define the transitive hull of density-connected points, forming density-based clusters.

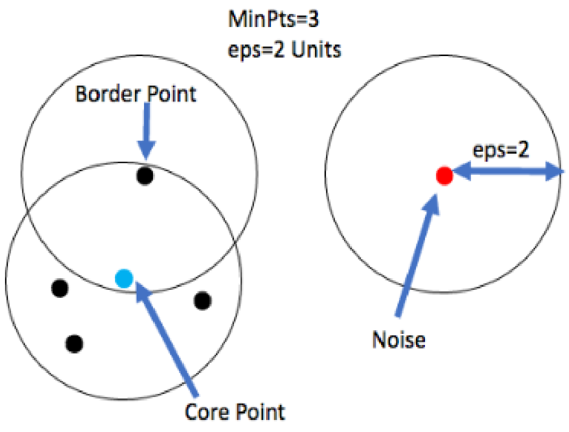


Figure 3.2: DBSCAN algorithm based on *eps* and MinPts [120].

The algorithm is described by:

1. Set of points to be regarded to form a graph.
2. Create an edge from each point c to the other point of the neighborhood of c.
3. If the set of nodes N contains no center points, then terminate N.
4. Select a node X that can be connected from c.
5. Repeat the procedure until all center points form a cluster.

Ordering Points To Identify the Clustering Structure (OPTICS) [121] The difficulty of finding density-based clusters with widely differing densities has also motivated hierarchical procedures for computing clusters at different density levels in a single pass [122]. Due to the fact that the connected components of different density levels are either disjoint or the cluster of higher density is entirely contained within the lower density cluster, the result of such hierarchical algorithms can be represented as a tree. Practical approaches for density-based hierarchical clustering include OPTICS [123]. OPTICS is an extension of the DBSCAN algorithm that is also based on the same parameters as the DBSCAN algorithm. The algorithm is as follows:

1. Select a point from the set of points which is a center point if at less Minpts are within the base distance.
2. For each point c create an edge from c to another point with a center distance of c.
3. Select a set of nodes that contain center points as a cluster that extends from c.

3.5 Comparison Between Clustering Techniques

The comparison of various clustering algorithms are given in table 3.1. Under partitioned clustering method, k-means clustering dominates and is still the most popular clustering method. In k-means, even if an object is quite far away from the cluster centroid, it is still forced into a cluster and thus, distorts the cluster shapes [124] which produce lower quality of clusters. The final results of the clustering is heavily dependent on the initial centroids. The whole process is carried out several times with different initial parameters. For a data set of fixed size this might not be a problem. However, in the context of big IoT data, this characteristic of the algorithm leads to heavy computational overload [10]. The k-means algorithms have problems like defining the number of clusters initially, susceptibility to local optima and sensitivity to outliers, memory space and unknown number of iteration steps that are required to cluster. The fuzzy C means clustering is really suitable for handling the issues related to understand ability of patterns, incomplete/noisy data, mixed media information, human interaction and it can provide approximate solutions faster [125].

The main motivations of BIRCH lies on two aspects, the ability to deal with large data sets and the robustness to outliers [126]. Also the BIRCH can achieve a computational complexity of O(N) where N is the number of objects. ROCK not only generates better quality clusters than traditional algorithm, it exhibits a good scalability property [107]. BIRCH and CURE both handle outliers well but CURE clustering quality is better than that of BIRCH. On the reverse, in terms of time complexity, BIRCH is better than CURE as it attains computational complexity of O(N) compared to CURE O(N^2 log N). The ClIQUE algorithm combines the advantages of density and grid methods. It divides data not only based on the grid, but also takes density into account. It partitions data into dense and sparse sets and focuses on dense grid cell data. However, CLIQUE algorithm divides each dimension equally according to the user setting. It may lead to a cluster being divided into several artificial clusters. In addition, the number of connections will grow exponentially and the computational complexity will be very high at high-dimensional data sets [86]. The performance results show that MAFIA is 40 to 50 times faster than CLIQUE due to the use of adaptive grids. MAFIA introduces parallelism to obtain a highly scalable clustering algorithm for large data sets [116]. DBSCAN (density-based spatial clustering of applications with noise), which discovers clusters of arbitrary shapes and is efficient for large spatial databases [125]. OPTICS find clusters of fixed density. It ensures good quality clustering by maintaining the order in which the data objects are processed, i.e., high density clusters are given priority over lower density clusters [116].

Table 3.1: Comparison of various clustering algorithms

Algorithm type	Algorithm name	Time of complexity	Suitable for large scale data	Suitable for high dimensional data	Advantages	Disadvantages
Partitional	k-means [104]	$O(N)$	Yes	No	This is one of the most useful clustering algorithm [125].	-There is no efficient and universal method for identifying the initial partitions. - It is sensitive to noise. - It can produce lower quality of clusters [10] .
	FCM [105]	$O(N)$	No	No	-The object could belong to all clusters with a certain value of membership. -Giving descriptions of objects in clusters in more detail [127].	FCM suffers from initial partition dependence as well as noise and outliers like k-means.
Hierarchical	BIRCH [106]	$O(N)$	Yes	No	-Robustness to outliers [106]. -Suitable for large databases [109].	-It can achieve a computational complexity of O(N). -Handles only numerical data. -Sensitive to order of data records [128]. -Inappropriate to high dimensional datasets. -Unable to detect arbitrary shape cluster.
	CURE [107]	$O(N^2 logN)$	Yes	Yes	-Robustness to outliers .It is proposed to address the big data challenge, it uses a random sampling technique to reduce the computational cost [86].	It includes both the hierarchical part and the divided part, which overcomes the disadvantage of using a single clustering center tend to discover spherical clusters [129].
	ROCK [108]	$O(N^2 + N_{m_m m_a} + N^2 \text{ logN})$	No	No	-It use a random sample strategy to handle large datasets. -Exhibit good scalability [107].	It cannot handle differing densities [129].
Grid	STING [110]	O(k)	No	No	It is highly scalable and can handle outliers well	It is not suitable for high dimensional data set.
	CLIQUE [111]	O(Ck + m k)	No	Yes	It automatically finds subspaces of high dimensional data space that allow better clustering than original space.	The accuracy of the clustering result may be degraded at the expense of simplicity of the method CLIQUE.
	MAFIA[112]	O($c^p + p^N$)	No	No	Adaptive grids have been proposed for rapid clustering of subspaces clusters [116]	Inappropriate to high dimensional datasets.
Model based	SOM [113]	O($N^2 m$)	No	No	It easily detect the noise . It able to manage missing data values.	-It is not suitable for high dimensional data set. -Generate a sub optimal weights that are not chosen properly.
	COBWEB [114]	O(N^2)	No	No	It can achieve high predictability of the nominal values of the variables given a cluster [125].	It is not suitable for high dimensional data set.
Density Based	DBSCAN [119]	O(NlogN)	Yes	Yes	-Discovers clusters of arbitrary shapes. -Efficient for large spatial databases [125].	It fails to find clusters of varying density due to fixed eps.
	OPTICS [121]	O(NlogN)	Yes	Yes	-Find clusters of fixed density .	Less sensitive to outliers.

In table 3.1, N is the number objects, m is the number of initial sub-clusters produced by the graph partitioning algorithm, m_m is the maximum number of neighbours for a point and m_a is the average number of neighbours for a point.

In the partitionel algorithms, the common criterion is relatively scalable and simple. It consists on finding the Euclidean distance between points and the center of the available clusters and assigning each point to the cluster with minimum distance [125]. However, these algorithms include poor cluster descriptors. They lie on the user to specify the number of clusters in advance. They present high sensitivity to initialization phase, noise and outliers. They are unable to deal with non-convex clusters of varying size and density [125]. In addition, they give bad result caused by the overlapping of data points [130].

Hierarchical method is based on the distance between objects and clusters. The idea of hierarchical methods is that objects are more related to nearby objects rather than the farther objects. The major problems which commonly occur in Hierarchical clustering algorithms are [128]:

- No objective function is directly minimized.
- Sensitivity to noise and outliers.
- Difficulty in handling different sized clusters and convex shapes.
- Difficulty in breaking large clusters.

The grid based clustering algorithms such as STING and CLIQUE has the ability to decompose the data set into various levels of details. The evolutionary approaches for clustering start with a random population of candidate solutions with some fitness function, which would be optimized [125]. The additional advantage is its fast processing time [131], no need of distance computations and easy to determine which clusters are neighbouring. The model-based method is hypothesized for each of the clusters and tries to find the best fit of that model to each other [116]. Model-based clustering algorithms are far less common than partitioned and grid based algorithms. Unfortunately, no implementation of model based algorithms is readily available which limits their usefulness in practice. In addition, they are often more computationally complex than comparable algorithms from the other categories [132].

In the density based clustering methods, the data space is composed of dense regions separated by regions of lower object density and a cluster is defined as a maximal set of density-connected points [125]. Density based clustering logarithms are used to form clusters of high quality with acceptable time complexity. They also give strategy to filter noise from real data. They are robust in finding clusters with different densities. They are suitable for high dimensional data and big data.

3.6 Other Big IoT Data Analytics Methods

In addition of clustering methods, several solutions are currently offered for the analysis of big data and big IoT data (Figure 3.3). These solutions need the same or higher processing speed than traditional data analysis with minimum cost for high volume, high velocity and high variety data [133]. These solutions are continuously developed to adapt them to the new developments in big IoT data. The exploration of IoT data has an important role in analytics and, like the clustering methods, the majority of the techniques are developed using data mining algorithms, based on a specific scenario, such as the prediction method, association rule methods and classification methods.

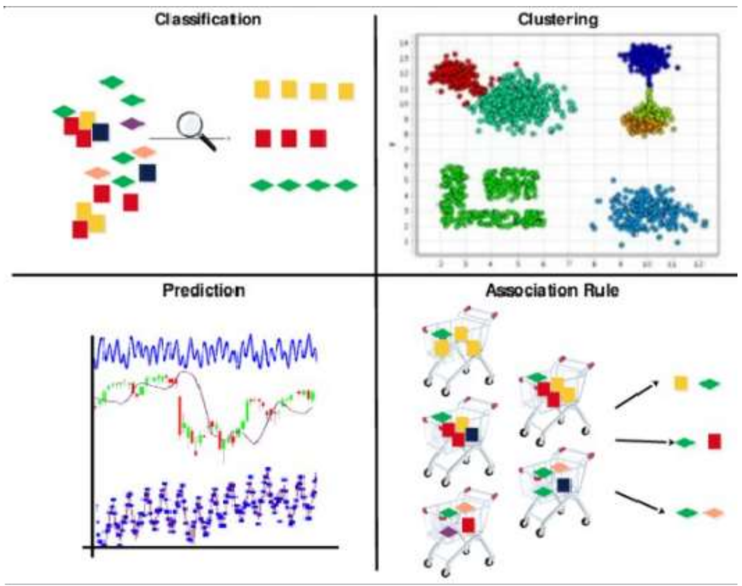


Figure 3.3: Big data analytics methods [85].

3.6.1 Prediction method

Predictive analytics employs the historical data, referred to as training data, to determine the results as trends or attitudes in the data.In big data analytics, processing demands are changed depending on the nature and volume of data. Rapid data access and exploration methods for structured and unstructured data are important concerns related to analysis of big data. In addition, data representation is an important requirement in big data analysis [85].

3.6.2 Association rule method

Associationn rule mining is centered on the identification and generation of rules based on the occurrence frequency for numeric and non-numeric data [134] . The data is processed in two ways The first way, sequential data processing, uses a priori algorithms, such as a priori algorithms, such as MSPS (Maximal Sequential Patterns by using multiple Samples) [135] and LAPINSPAM (LAst Position INduction Sequential PAttern Mining) [136], to identify interaction associations. The second way of processing the data according to the association rule is temporal sequence analysis, that uses algorithms to analyze patterns of events in continuous data.

3.6.3 Classification method

In supervised classification the class (label) of an object is predefined. The major objective of the classification approach is to develop a tool or algorithm, that can be used to predict the class of an unknown object, which is not unlabeled. This tool or algorithm is named a classifier. The objects in the classification process are more commonly represented by instances or patterns. A pattern consists of a number of features (also known as attributes). The classification precision of a classifier is evaluated by the number of test patterns it has classified correctly [125]. One of the ways of classification SVM (Support Vector Machines), is based on the theory of statistical learning to recognize patterns in the data and generate groups. In the same way, K Nearest Neighbor (kNN) is usually mechanisms for retrieving patterns from large datasets, so that the recovered objects are similar to the predefined category [137].

However to find unknown or hidden patterns is more difficult for IoT big data. Also, extracting precious information from big data sets to improve decision making is a very critical task. In additional there are usually huge amount of data produced in IoT applications, however, these data lack having labels, which makes these types of methods infeasible to be used in IoT environment [10].

3.7 Conclusion

Compared with data analytics methods namely prediction method, association rule method and classification method, clustering methods present interesting characteristics speciality DBSCAN (Density-Based Spatial Clustering of Applications with Noise) and OPTICS (Ordering Points To Identify the Clustering Structure) algorithms which have surpassed the k-means algorithm in term of homogeneity quality of clusters. The use of DBSCAN or OPTICS algorithms, as a pre-processing step of IoT data store, will result in the creation of clusters of high homogeneity in term of data type and dimension. These homogeneous clusters will reduce data overlapping which represents a serious challenge often encountered during big IoT data indexing in both metric and multidimensional spaces. Another advantage of these clustering methods is the possibility of introducing parallelism in both big IoT data indexing process and similarity query search.

4 Big IoT Data Indexing

4.1 Introduction

Indexing is a widely used method to store big IoT data and to provide fast answer when searching similarity queries. Indexing methods were proposed as solution of the challenge of processing and storing big IoT data so that the similarity query search proceed efficiently. This challenge came from the production, by various connected devices in IoT architectures, of different types of data in large volumes at very high speeds. In this chapter, we will present most indexing methods in both multidimensional and metric spaces (Figure 4.1). For both spaces, indexing methods will be grouped into two groups: centralized methods and distributed methods. At last, a comparative analysis of these methods in term of advantages and disadvantages will be presented. In this chapter, a particular interest is given to the indexing methods in metric space regarding their link with this work.

4.2 Multidimensional Space Indexing Methods

Indexing techniques in multidimensional spaces can be categorized into two main types depending on the structure: hashing methods and tree methods. The hashing methods are regrouped into Locality sensitive hashing methods (LSH) and

Learning to Hash methods (L2H) (Figure 4.2).

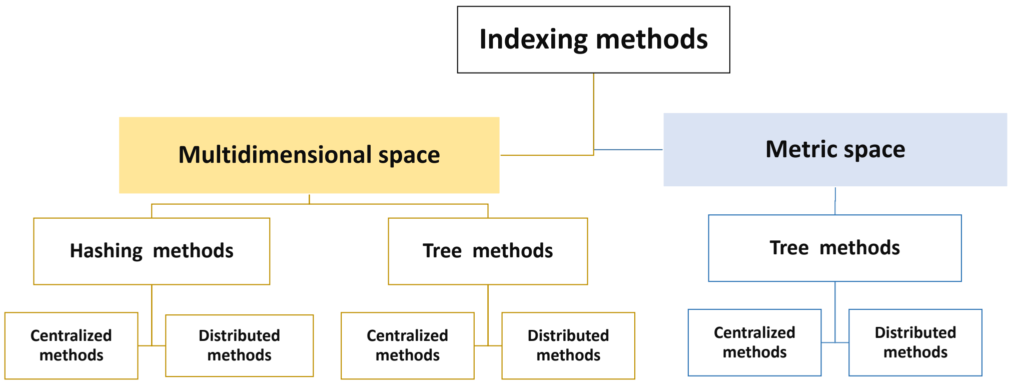


Figure 4.1: Global taxonomy of IoT data indexing methods.

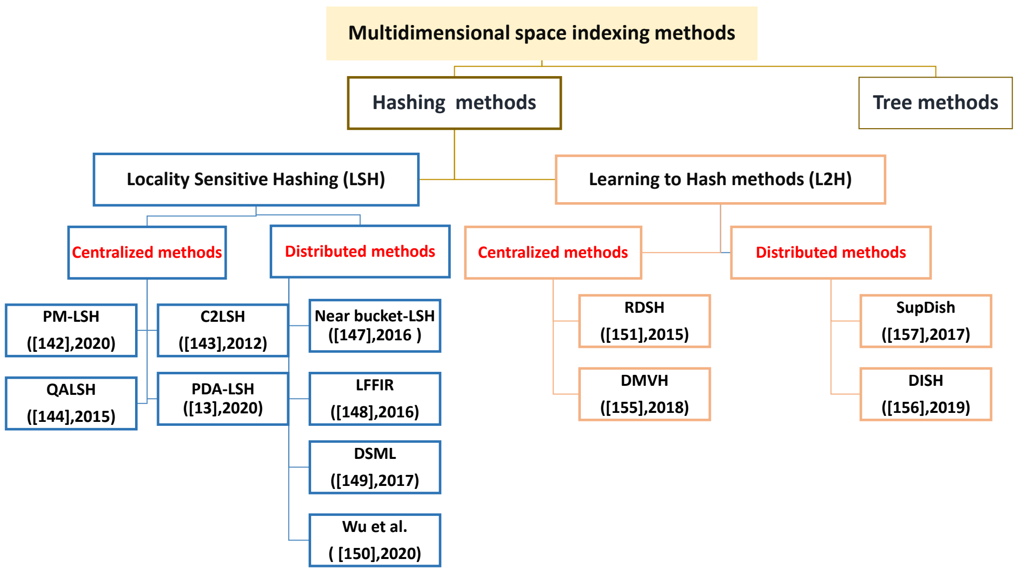


Figure 4.2: Hashing methods in multidimensional space.

4.2.1 Hashing methods

4.2.1.1 Locality Sensitive Hashing methods (LSH)

Hashing is an approach that consists in transforming the data object into a low-dimensional representation, or in an equivalent manner into a small code of bits. Locality Sensitive Hashing methods (LSH) and their variants are widely used methods [138][139][140][141]. The LSH scheme has been first introduced by Indyk et al. [138] to be applied in the binary Hamming space $\{0, 1\}^d$ and later extended to be used in the Euclidean space $\mathbb{R}^d$ by Datar et al. [139]. LSH maps the points in the data set to buckets in hash tables by using a set of predefined hash functions that are designed to be locality sensitive so that close points are hashed to the same bucket with high probability [142]. In this work, the presented LSH methods are regrouped into centralized methods and distributed methods.

4.2.1.1.a Centralized methods

- Collision Counting LSH (C2LSH) is a scheme which can ensure the quality of the query by selecting the size of the LSH function appropriately and the collision threshold dynamically [143].
- Query-Aware Locality Sensitive Hashing (QALSH) [144] utilized two techniques to improve upon accuracy. The first technique introduced query-aware hash functions by creating a B+-tree on each random projection. The second technique performing incremental range queries until top-k candidates are found.
- PDA-LSH (Projection Distance Aware LSH) is a locality sensitive hash method proposed to speed up the approximate c-ANN search with a low cost of index maintenance [13]. It is based on the use of the LMS-tree [145] which indexes the pairs (o_i, id) of ith projection of an object with their identifier.

- PM-LSH [142] in this method, the points are transformed into a low-dimensional space, called the projected space. The coordinates of a point in the projected space are the point's hash values. They use PM-tree to organise these points. However, this index presents the difficulty of estimating the original distance between the points after the results of a query are obtained. The storage space consumption by the hash tables. The complexity of the query search needs the computation of the hash function of a query in addition to the computation of probability on the candidate points.

The above mentioned methods use a number of hash tables that are necessary to ensure the quality of the search. However, because of the limitations of storage space and server processing ability, centralized indexing schemes are not feasible.

LSH enables sublinear search time in high dimension, but usually requires long hash codes. To generate compact codes, it is realized that hash functions should be adapted to data distribution because indexing schemes become impractical for large data objects [146].

4.2.1.1.b Distributed methods

- Near bucket-LSH this method, composed by Kraus et al.[147], integrated LSH and cosine similarity metrics in a P2P Content Addressable Network (CAN) architecture to enhance network efficiency when searching near buckets.

- LSH-based Fusion Features for Image Retrieval (LFFIR) Liao et al.[148] introduced a distributed image retrieval framework for similar search content, which can effectively integrate image retrieval based on multi-functional in the Chord P2P network into a cloud data center.

- Decentralized Search for Large and Mobile wireless networks (DSLM) is a proposed system for large mobile wireless networks, where it divides the entire network in to smaller regions. Allowing nodes to join, leave, distribute metadata or make requests. Next the LSH function is used to distribute similar group documents to the same region. Then a geographic routing method based on the region is applied to route messages messages between nodes [149]. However, the LFFIR and DSML indexing schemes insufficiently account for the load balancing problem, that is one of the key issues on the overall performance of the distributed system [150].

- A hashing method to obtain a balanced distributed p2p network is proposed in [150]. The concept of virtual node is used to adapt to dynamic changes in data load and network environment.

Nevertheless, the LSH family needs a longer code length to ensure search performance, which leads to a higher storage cost, thus limiting the scalability of the overall algorithm [151]. In additional, as LSH originally was developed to find objects in a fixed radius, to guarantee the quality of the intended to search for objects in a fixed radius, to unsure the quality guarantee, it needs to construct indexes for different radii. Thus, in this case, hundreds or thousands of hash tables are constructed, which results in high space and search costs [142].

4.2.1.2 Learning to Hash methods (L2H)

To minimize the search cost in LSH methods, Learning to Hash methods (L2H) have been proposed for their ability to learn similarity by preserving hash functions adapted to a given data set. These methods are classified into four categories according to the degree of supervision, namely: unsupervised hashing, supervised hashing, semi-supervised hashing and deep hashing [152]. According to the literature [153] [154], L2H methods adopts Hamming ranking (HR) as a query technique that probes the buckets in ascending order of their Hamming distance to the query.

4.2.1.2.a Centralized methods

- Dynamic Multi-view Hashing (DMVH) [155] capable of adaptively increasing hash codes according to dynamic changes in the image. These hashing techniques also use multi-view features to achieve more efficient hashing performance.
- Robust Discrete Spectral Hashing (RDSH) [151] is a hashing approach to facilitate large-scale semantic indexing of image data. It is aimed at learning a set discrete binary codes and robust hash functions within a unified model. This approach is not adequate for a large and dynamic databases [152].

4.2.1.2.b Distributed methods

- Distributed Indexing by Sparse-Hashing (DISH) is a distributed kNN index for cloud-based systems, based on sparse hashes [156]. It can help to overcome challenges related to large-scale index distribution associate vectors to several index nodes based on their orthogonal similarities and search for large-scale distributed images. DISH supports distribute documents and queries in a balanced and redundant way between nodes [156].
- Supervised Distributed hashing (SupDisH) is an efficient method that learns discriminative hash functions by taking advantage of the semantic information of the labels in a distributed manner [157]. The distributed hash problem is discussed in the context of classification, where it is expected that the learned binary codes are distinct sufficiently for semantic retrieval [157].

LSH methods operate with the predefined hash functions regardless of the underlying dataset, where L2H learns custom hash functions based on the dataset. While there is an additional training step necessary, some studies have shown experimentally that L2H outperforms LSH in terms of query efficiency [158], [159],[160]. However, the Hamming distance is a gross indicator of the similarity between the query and the elements of a bucket as it is discrete and has a limited number of values. Consequently, Hamming ranking may not define a good order for buckets having the same Hamming distance from the query. As a consequence, HR generally probes a large number of unfavorable number of adverse buckets, leading to low efficiency. A solution is to employ a long code so that the Hamming distance can classify buckets into larger categories. However, the long code has challenges such as time-consuming sorting, high storage demand and low scalability, in particular for large-scale datasets [161].

4.2.2 Tree methods

4.2.2.1 Centralized methods

The centralized indexing techniques, in the multidimensional space, can be classified in two major approaches: the space partitioning, which uses space cells to index the data, and data partitioning which uses cells of similar objects (approximation function) to index the data (Figure 4.3).

4.2.2.1.a Space partitioning methods

Several previous studies have focused on indexing through tree structure that relies on the successive division of space. In the multidimensional space. Among these methods of space partitioning, we can cite the following:

K-dimensional tree (Kd-tree) is a static method based on the partitioning of space into K dimensions [162]. It is based on the division of a dataset into two equal subspaces $(Left, Right)$ by the median m of the dataset.

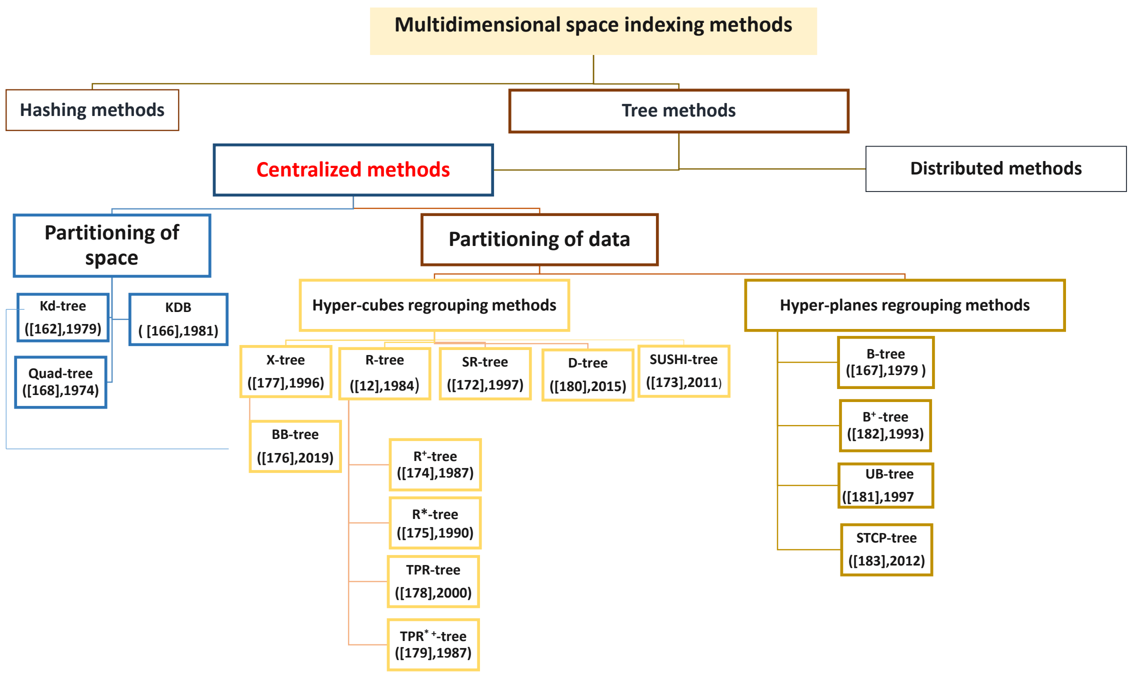


Figure 4.3: Centralized tree methods in multidimensional space.

This process of dividing the spaces repeats recursively. The data is structured in the form of a binary tree. The principal disadvantage of the Kd-tree is that it is unbalanced since the hyperplane space divider does not partition planes in the best place. This creates overlaps between neighboring regions, and this causes the cost of I/O operations to be higher [163]. Additionally when searching for a kNN query in high-dimensional spaces, most of the points in the tree will be traversed and the efficiency is not better than with an exhaustive search [164], [165]. On the other hand, partitioning the space using hyperplanes in the Kd-tree in situations where the query point is close to the boundary between two neighbor regions. It is necessary to visit the two neighbor regions, This affects the response time negatively [162], [166].

KdB-tree [166] is a combination of Kd-tree [162] and B-tree[167]. It is a dynamic structure and balanced tree. It is proposed to improve I/O performance of Kd-tree [162]. The KDB-tree can't ensure minimal storage consumption. It only considers point in time data and insufficient search performance.

Quad-tree [168] partitions the two-dimensional space into quadrants and consists of many quadrants and comprises various partition index spaces. It is not balanced as is not selecting the optimal division of the space as a Kd-tree . Moreover, Quad-tree ignores the distribution of the data in space partitioning process [169].

4.2.2.1.b Data partitioning methods

This approach is based on the partitioning of data by the way of packages grouping data, also called "enclosing forms". These methods can be classified into two classes, the first one based on the grouping of objects into rectangles of minimum delimitation (hyper-cubes). The second class are the methods that are based on the grouping of objects in regions of minimal delimitation (hyper-planes).

Hyper-cubes regrouping methods

1. R-tree proposed first by Guttman [12]. It is considered as one of the first methods that indexes data in multidimensional spaces in the form of a balanced hierarchical partitioning into sets of rectangles called Minimum Bounding rectangles (MBRs). It is a spatial access method used to index geographic coordinates. Minimum Bounding Rectangles (MBR) is determined by a pair of vectors where the components of the first vector are two less than or equal to those of the second vector. This pair of coordinates defines the smallest volume that encloses a given set of points and/or geometric forms. R-tree is efficient structure for range queries [170] dynamic [12] and balanced

[171] However, it suffers from the problem of overlap between the rectangles, which leads to a difficulty in finding the objects in these rectangles. In higher dimension the time and the complexity of the computations is augmented [172]. In additional it rapidly degrades for higher dimension [173].For its improvement, many methods have been proposed. R^+-tree [174] eliminates the overlapping rectangles by their dividing until all overlaps are eliminated however, it causes the increase of the tree height. R^*-tree [175] minimize the overlap of rectangles by inserting a few child nodes before dividing a node. It improves partitioning by aggressively reinserting data objects leading to a more efficient search performance [176].

2. eXtended node tree (X-tree) [177] is an additional multidimensional index that is similar to the R-tree which aims to limit the problem of overlapping forms. It adopts a completely different strategy for partitioning nodes, which are extended with variable sizes, called extended nodes.It is enhances the R^*-tree by introducing overlap-minimizing splits for the objects that caused the overlap. Eventually degenerating to a sequential scan [173].

3. Sphere and Rectangle (SR-tree) [172] is based on the grouping of objects into regions, where each region is the intersection of a hyper-rectangle and a hyper-sphere. The idea is that intersections of these shapes give small areas, which avoid overlapping. However, this solution is very complex to build these shapes and to find the intersection areas.

4. Subspace based High-dimensional Indexing (SUSHI-tree) [173] is proposed to index high dimensional objects. The space is divided into subspaces by clustering. The internal nods are clusters defined by a list of upper and lower bound values for the relevant dimensions of the cluster and a pointer to its corresponding child node representing the objects. However, in this index

the division of the space into subspaces does not guarantee that each cluster contains objects of the same type.

5. Time Parameterized R-tree (TPR-tree) [178] is a variant of the R*-tree for processing movement data. It supports queries for present and future positions of moving objects. Moving objects are enclosed in a bounding box that does not shrink. The position and period of an object are implemented by a function.

6. TPR*-tree [179] is an improvement of TPR-tree. It added insertion and deleting to the TPR-tree.

7. Decomposition Tree (D-tree) [180] is a virtual tree without internal nodes used for the indexing of the multidimensional motion. Which are replaced by an encoding method based on integer bit-shifting operation.

8. Bubble Buckets tree (BB-tree) [176] is based on the combination of the Kd-tree structure [162] and X-tree [177]. It recursively partitions the data space into k partitions, its leaf nodes store objects in elastic buckets named Bubble Buckets (BB). Each BB contains m-dimensional objects. Similarity search queries are not considered in the structure.

Hyper-planes regrouping methods

1. B-tree [167] in a binary tree, each node of order d contains at most $2d$ keys with $2d + 1$ pointers. The search in a B-tree depends on the branch from the node to the query key. When the query is less than the saved key, the left branch is chosen, if the key is greater, the right branch is chosen. The drawbacks need a linear space for storage and logarithmic time for the basic operations of insert and find. In addition, the B-tree only works well for one-dimensional data [181].

2. B$^+$-tree [182] tree is an m-ary tree that has a varied and usually large number of children per node. A B$^+$-tree consists of a root, internal nodes and leaves. The root can be either a leaf or a node with two or more children. Many indexing schemes are based on the B$^+$-tree.

3. STCB-tree [183] indexes the trajectories of the motion in the past, the present and anticipates the future.

4. UB-tree (Universal B-tree) [181] is an improvement of the B-tree. It organizes the objects in an n-dimensional space (called universe) so that they can be stored, managed, retrieved from and deleted from from and deleted from peripheral storage very efficiently.

4.2.2.2 Distributed methods

The distributed methods, cited in what follows, are grouped in figure 4.4.

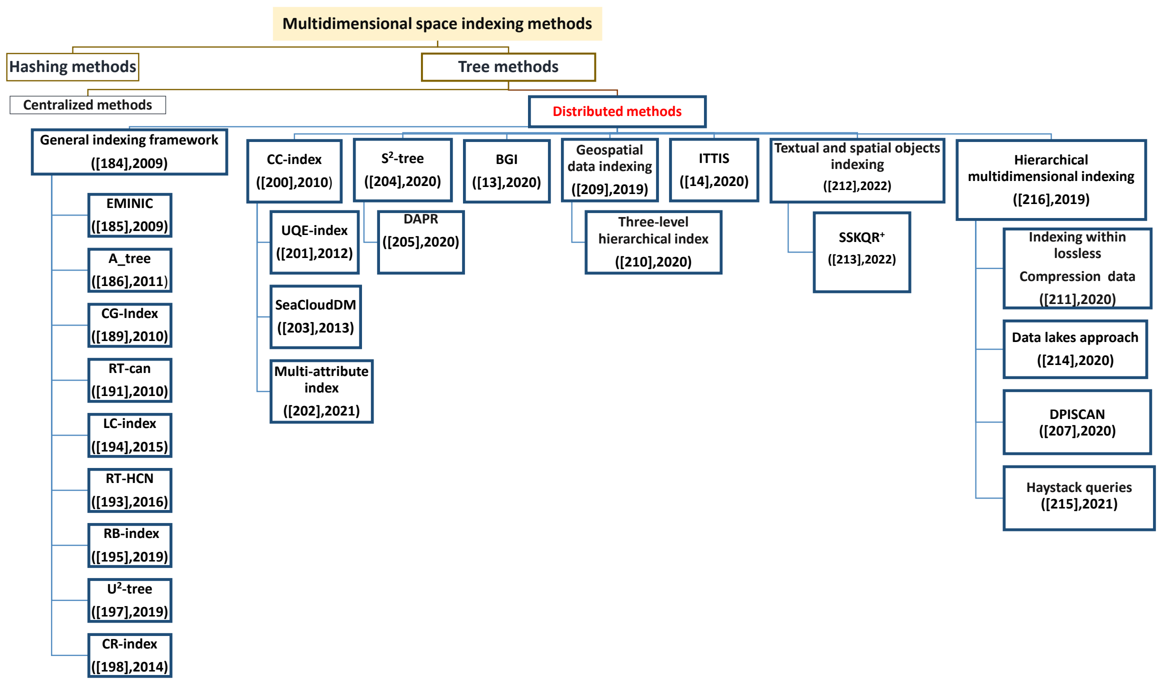


Figure 4.4: Distributed tree methods in multidimensional space.

General indexing framework is the first index that was presented for indexing data in the cloud system based on the overlay network [184]. This index has two layers, with the servers organized in the overlay network, thus every server constructs their own local index for accelerating retrieval of data. The global index is constructed over the local index by choosing part of the local index and publishing it to the network. The global index is used to provide an overview of the local index. While this index scheme is scalable and flexible, the Peer-to-Peer structure is not well suited for cloud systems [185].

A-tree proposed, first, by Papadopoulos et al.[186], is the appropriate method for cloud computing environments. It is a distributed and scalable indexing scheme for multidimensional data, capable of handling both point and range queries. It is based on the combination of R-tree [12] and Bloom filters [187]. It is only used for multidimensional data.

Efficient Multi-dimensional Index with Node Cube (EMINC) [185] is a multi-dimensional two-layer index. It provides fast query processing and efficient index maintenance. It is an approach for indexing large IoT datasets: a hierarchical approach to building a multidimensional index for a cloud system. It combines R-tree [12] and Kd-tree [188] for data organization.

CG-Index [189] is a two-layer index constructed over the BATON network [190] and it employs B-tree to address one-dimensional high speed queries.

R-Tree based index in CAN (RT-can) [191] is a multidimensional indexing scheme in data centers. RT-CAN combines the CAN-based routing protocol (Content Addressable Network) [192] and the R-tree-based indexing scheme [12] to address efficient multidimensional query processing in a cloud system. RT-CAN organize storage and computation nodes in an overlay structure based on an extended CAN protocol. The RT-CAN index is constructed on top of local R-trees and published on the cluster servers. They use a method to mapping a selected R-tree node to a CAN node. However, RT-CAN is constructed in a p2p network, with nodes dispersed largely in the real world and unstable connections between nodes, resulting in unreliable services [193]. RT-CAN is not scalable regarding the dimensionality of the data. The original overlay network must be expanded and additional servers need to be added to reconstruct the index, thus costing. The query processing algorithms are designed to support point, range and KNN queries for the RT-CAN index.

Local and Clustering Index (LC-index) has been proposed by Feng et al. [194] as a combination of the RT-CAN [191] index and the CG index [189] what enhanced the flexibility and the insertion of multidimensional range queries. This index is dynamic and supports the operations of insertions and deletions however, it presents a high cost of storage.

Hierarchical Irregular Compound Networks (RT-HCN index) Hong et al.[193] proposed an indexing scheme that integrates R-tree [12] and a routing protocol based on Hierarchical Irregular Compound Networks (HCN). This scheme, called RT-HCN is proposed to organize storage and compute nodes in an HCN overlay, in server-centric cloud storage system. The RT-HCN is composed of two layers. In the first layer, the data is distributed in different servers and locally indexed using R-tree. In the second layer, the local indexes are distributed across servers as a global index. Although R-Tree is a balanced and dynamic tree, the search process degrades when the index data is large due to the fact that R-Tree has the overlapping multiple MBR regions.

RB-index [195] is an efficient and scalable multidimensional indexing scheme for the BCube topology [196] in modular data centers. RB-Index is a two-layered indexing scheme that integrates the BCube-based routing protocol and the R-tree-based indexing structure. It use routing tables of a set of switches, in order to build an indexing space with n dimensions. This space is divided into n subspaces. Each server creates its R-tree and then publishes its address and its MBR in its table of routing. In RB-Index, they are building several distinct indexing spaces with selected dimensions according to the query history. Every server takes over part of the indexing space according to a mapping scheme. According to the authors, the division of the space into several sub-spaces during the publication of the R-tree nodes in the form (ip, MBR) produces false positives.

U^2-tree Gao et al. [197] proposed a universal two-layer indexing scheme built on cloud storage system with tree-like DCN (Data Center Networks) topologies called U^2-tree. The first layer, named local index, facilitates the query processing on local hosts. The second layer, named global index, locates the host in which the data is stored. The construction of the U^2-tree starts with the build of the local index by using the B^+ tree [182] for local data to efficient query search. The global index indicates in which local host the data is located. Each host will manage a portion of the global index within a certain range. The U^2-tree support point query, range query, and kNN query search however, the cost of the update and the maintenance of the distribution index is high.

Continuous Range Index(CR-index) Wang et al. [198] proposed a continuous range Index (CR-index) for indexing observed data based on its value ranges and type attribute. CR-index builds a compact indexing scheme where a measurement data items and the observation data items are aggregated into boundary blocks based on their interval blocks. The indexes are built to answer range queries.

However, this approach is only able to index data with a unique dimension [199]. In additional the unique dimension in the CR-index made it useless for data of higher dimensions [198].

Complemental Clustering Index (CC Index) [200], is an additional index based on Key-value store. A secondary index table has been built for each indexed column. In order to improve the random readability, more detailed information of each record has been pushing into the secondary index table, thus the random reading might change to a sequential reading. Also, the author suggested some methods of optimization to support multidimensional queries. CC Index is simple to implement, however, it suffers from various drawbacks. firstly, it requires a large amount of additional storage space when there are many indexed columns, secondly CC Index does not support adding not support adding or deleting indexes after the table has been built.

Update and Query Efficient index framework (UQE-Index) Ma et al.[201] proposed an efficient update and query index framework (UQE-Index) based on a key-value store that can support a high insertion rate and simultaneously provide an efficient multidimensional query. The UQE-index divided data into two types: historical data and current data, which were indexed with different granularities. For the historical data, a finer combined index was applied. A spatial index was built inside a temporal index. For the current data, in order to handle high updating pressure, a coarse-grained index was applied They used data partitioning and tree-based indexing to develop the HBase-based UQE-Index framework to make data management more efficient. According to them, the response time under the UQE index is lower than that of the EMINC framework. However, this framework supports only range query. The kNN search query, which is more generalized than the range query, was not tested. The UQE-index has proposed a complete index structure that deals with spatial-temporal attributes, there are other attributes in the IoT domain [202].

SeaCloudDM [203] the continuous data generated from IoT devices is being received, stored and processed in a sea-computing layer. The products of the sea-computing computing layer are numeric key sample values that are considerably smaller than the original data from the devices. This key sample data is passed to the cloud data management layer for later processing. Relational Data-Base and Key-Value (RDB-KV) store combined cloud data management model is employed to manage SQL queries and keyword search. However this method manages massive data from heterogeneous sensors in the cloud, which suffers from latency problem.

Multi-attribute index [202] in this approach, four types of attributes are employed: spatial, temporal, keyword and value. A specific indexing method is allocated for each attribute and the inclusion of these four indexes in a combined index needs a certain sequencing that determines the performance of the query search. The query search performance is improved by taking into account all possible sequences and by automatically determining the most efficient combined index for each query. This approach focuses on enhancing the performance of queries search and authors do not specify the way to store indexed IoT data [202]. The B^+-tree is used for storing the temporal attribute and value attribute.The R-tree is used for store spatial attributes, That are usually used to describe the geographic location of IoT data. Since B^+-tree is a balanced structure and the it can support range queries efficiently. However the authors, only considering numerical data which is one-dimensional data.

S^{2}R-tree [204] which integrates spatial and semantic information. It adopts two layers. The first is a spatial layer used R-tree to group objects according to their geographical coordinates. The second is a semantic layer, transforms the high dimensional semantic vectors to a low dimensional space.

Distributed Access Pattern R-tree (DAPR-tree) [205] for spatial data retrieval in a distributed computing environment. The balance of the index structure and parallelization of the workload between several main computation nodes allows rapid data recovery. For this reason, the authors apply the R-tree structure on a three-tier distributed environment: the principal tier is the input of the global index and manages the data partitions for the sub-tier. The sub-tier constructs a number of sub-tree indexes for different data partitions, this sub-tree adopts an R-tree, R-tree * or APR tree. A data and computational tier provides data and computational resources to take care of the operations of the DAPR tree. During the search for a given query, the master node sends the query to all partitions at the same time. Each partition searches locally. Then all partitions send their results to the master node. DAPR is an efficiently indexing approach for spatial data retrieval in a distributed environment, it assures the balancing of data distribution, workload and data access. However this tree is limited to applications that have relatively stable data access patterns. It is not adapted to a dynamic IoT environment. In addition, the master node can overload when multiple queries arrive. moving object

Block Grid Index (BGI) Yang et al. [13] proposed this index which is a distributed method for large-scale moving objects with two-layer: grid-based index and DBGKNN (a distributed k-nearest neighbours query search algorithm based on BGI). According to the authors, this work requires an optimization of the index structure by the incremental update of the kNN query search when the objects are moving. Moving objects represent the only type of data that the BGI method can index. temporele data

In-memory based Two-level Index Solution in Spark (ITTIS) is another framework for processing temporal data in a real-time distributed system based on Apache spark memory [14]. ITTIS consists of three levels: the first level is the partition unit, which is responsible for partitioning all temporal data into distributed nodes. Each partition consists of a set of intervals. Each interval is defined by (start and end, value). The second level is the local index unit, every partition, is indexed by MVB-tree (MultiVersion B-tree) [206]. The third level is the global index unit, which is located in the master node. It is used to collect the intervals of all partitions in the master node of the spark Apache. They built the BST (Binary Search Tree) for all these intervals. The search for a query is done in two steps. The first step is to find the candidate partition. The query search is achieved in the BST tree by pruning the sub-trees that are not suitable. In the second step, the search begins in the candidate partition to find the searched record. It provides native support for querying big data. However, dividing the research into two stages is that it can take a long time. In additional, this framework only supports temporal data and a specific type of queries that can not be replaced by other types of IoT data.

Distributed and Parallel architecture with Indexing for structural clustering using SCAN algorithem (DPISCAN) Kumar et al.[207] proposed this approach to thread-level parallelism on the Apache Spark distributed architecture. A cache-based indexing technique creates indexes for the neighborhood vertices using CSS-tree [208] structure to take care of a combination of different threshold values. This work focuses only on data indexing and no query search method was proposed. geospatial

Geospatial data indexing In [209], a geospatial data indexing was performed in the cloud where a parallel R-tree [12] and its parallel variants were constructed. Three construction methods were used: Apache Spark in-memory, Apache Spark on disk and MapReduce. Each one is looking for the fastest way in building, updating and executing spatial query. One of these three methods, the Apache Spark in-memory, reduces significantly the time for indexing geospatial data and querying ranges. However, this method is only used for geospatial data where the dimension is limited to three.

Three-level hierarchical index Hu et al. [210] presented a three-level hierarchical indexing method to enhance Apache Spark and the Hadoop Distributed File Storage System (HDFS) for managing data from Earth observations and model simulations. They combine the global kd-tree index for the master node with the local hash table for each data node, which provides a scalable indexing strategy for searching large raster geospatial data in a distributed environment. They developed a data distribution strategy to address query parallelism while maintaining high data locality. This method only supports querying large geospatial data and is not tested for other data types.

Indexing within lossless compression Data Doan et al. [211] proposed an indexing model consisting of a lossless compression technique for IoT data as well as the benefits of bit-padding, bit-blocking, and Huffman coding. It minimized the data size during the compression, which does not require fixed 8-bit streams. The index is based on timestamps that supports access to compressed data without full decompression. which is linked in during the compression process. This framework focused on building indexing within lossless compression for floating point time series data. According to the authors, this framework needs to be enhanced by addressing temporal alignments and de-duplication problems when IoT streaming data is sourced from multiple devices. However, the use of indexes based on timestamps made this method adapted for a specific type of data [211].

Textual and spatial objects indexing Bavirthi et al. [212] proposed an indexing mechanism combining textual and spatial objects for skyline querying. An inverted file is used for indexed textual objects and attached to the R^*-tree. However, adapting skyline queries in a dynamic environment like IoT is a difficult process when removing and inserting tuples at any time or in specific time intervals.

SSKQR$^+$ Recently, other framwork in the literature are introduced to recover the most relevant data [213]. They introduced framework for spatio-textual skyline querying with R^+ tree indexing technique. To relate keywords provided by the user and the geometric data of the user with an efficient sky R$^+$ named SSKQR$^+$. The skylines of data with geometric information are recovered by searching the geometric data points closest to the user's location. The recovered skylines are also verified by specifying a threshold value 'k' named top-k skyline querying. However, the use of the R^+-tree is much difficult during construction and maintenance. In addition this framework has no consideration for the IoT environment.

Data lakes approach According to the Seattle Database Research report [214], a new approach called data lakes is proposed to store and analyze a huge amount of data. In this approach, data is flowed to a distributed storage system such as HDFS where they are analyzed and managed instead of uploading them to data warehouses which induced high maintenance cost [215].

Haystack queries The approach, proposed by Weintraub et al. [215], aims to optimize needle in a haystack queries in cloud data lakes. This approach consists on the construction of an index structure that maps indexed column values to their files. Parallelism is used, in this approach, for ensuring the scalability in both compute and storage senses. The cost of this index is significantly high and the data load time is much longer.

Hierarchical multidimensional indexing A hierarchical multidimensional indexing method based on binary space partitioning (BSP) was proposed by Wan et al. [216] for efficient spatial query processing. After evaluating k-d-tree, quad-tree, k-means clustering and Voronoi diagram data structures, they found that the Voronoi diagram data indexing method is suitable for general query operations with a response time of $O(log(n))$. However, the dimension limitation and the specific type of query make this method difficult to generalize.

4.3 Metric Space Indexing Methods

The characteristics of IoT data, the diversity of type, format and dimension require us to consider a metric space. Several benefits of searching in a metric space are available. The most important is that a larger number of data types can be indexed, as this approach is based only on the calculation of distances between objects and not on their content [15].

4.3.1 Centralized metric space indexing methods

Centralized indexing techniques in metric spaces are classified into two main approaches. The first approach partitions the space and it is divided into two concepts: the hyperplane partitioning and the ball partitioning. The second approach partitions the data (Figure 4.5). In the following, some centralized techniques, classified according to the two above mentioned approaches, will be presented.

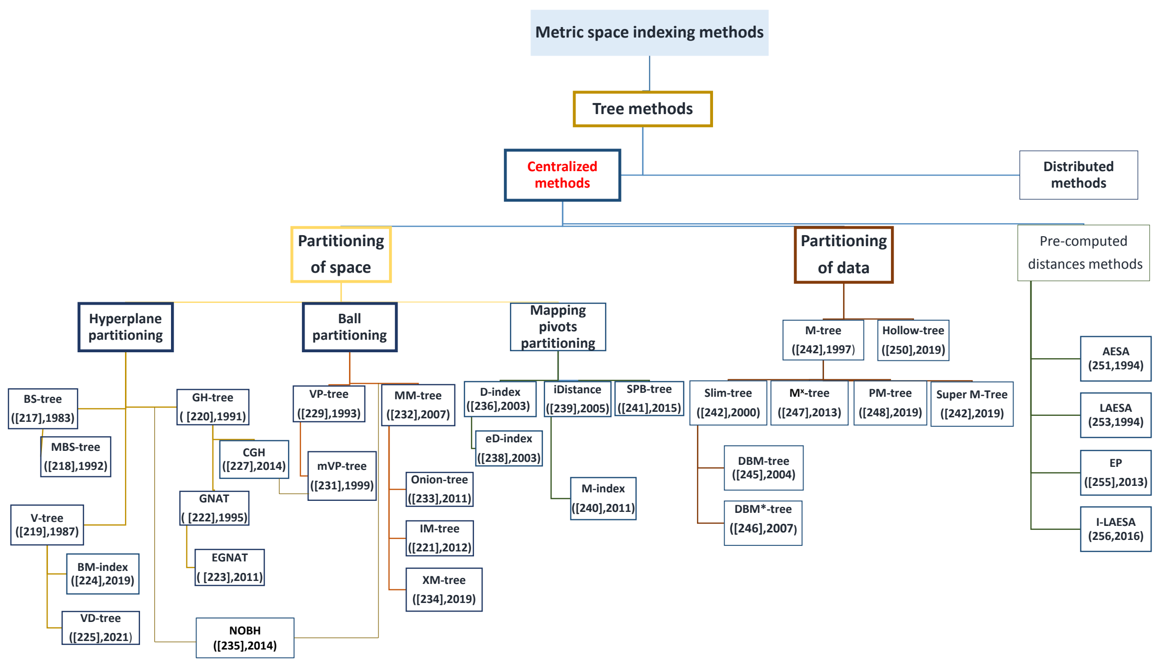


Figure 4.5: Centralized metric space indexing methods.

4.3.1.1 Space partitioning

Indexing techniques, based on space partitioning, are grouped into two indexing methods based on the partitioning concept: indexing methods based on the hyperplane partitioning and indexing methods based on ball partitioning. Other indexing methods, proposed to reduce the computation of distances in a metric space, will be also presented. In these methods, the metric space is partitioned into vectors by mapping functions.

Hyperplane partitioning

- BiSector tree (BS-tree) [217] is a recursive binary tree constructed basing on the generalized hyperplane partitioning. The coverage radii of each pivot are determined and stored in the corresponding nodes. The radius of coverage represents the maximum distance between the pivot and all objects in its subtree. The type of search in this index is range query only.

- Monotonous BiSector tree (MBS-tree) is a modification of the BS-tree proposed by Noltemeier et al. [218] in order to minimize the cost of computing distances when searching for a query range. The pivots in the nodes of the tree are minimized so that the pivots corresponding to the left subtree and the right subtree are copied in the corresponding inner child nodes, respectively.

- Voronoi tree (V-tree) [219] is a ternary tree with each node representing at least two and at most three points of $\mathcal{O}$. The root node is allowed to represent only one element. The V-tree is unbalanced structure. If a new leaf v has to be created in order to store a new point P and Q is P' nearest neighbor of of the (three) points stored in the father node of v, then Q (redundantly) has to be stored in v too. The insertion of new objects into new leaves of a V-tree (for example objects O_4, O_5 and O_6 (Figure 4.6)), induced a new space partitioning in Voronoi diagram.

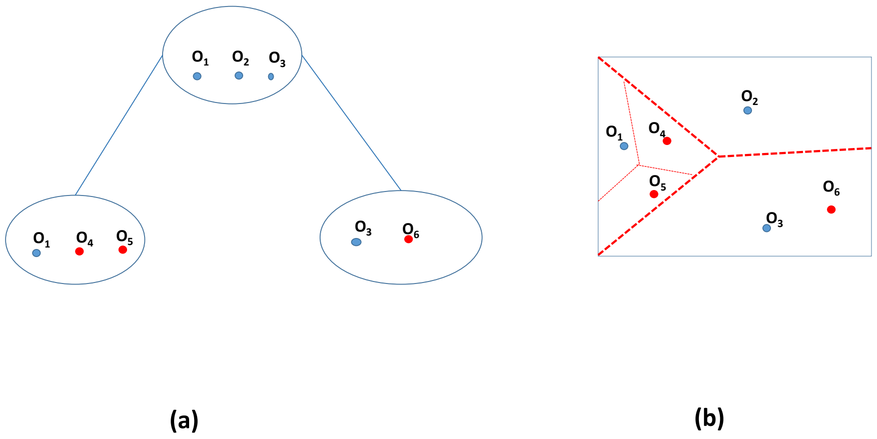


Figure 4.6: (a) Insertion of new objetcs in the V-tree. (b) Corresponding space partitioning in Voronoi diagram [219].

- Generalised Hyper-plane tree (GH-tree) [220] is similar to BS-tree since both partition the dataset recursively via the generalized hyperplane principle. The distinction is that Generalised Hyper-plane tree (GH) uses the hyperplane between the pivots p_1 and p_2 to determine the subtrees and not using covering radii as a pruning factor in the search process. The two points p_1 and p_2, chosen randomly, partition the space into two regions. The other objects are associated to their closest pivot p_1 or p_2 and thus, a generalized hyperplane that separates the dataset into two subsets is created (Figure 4.7). The space complexity of GH-tree and BS-tree is $0(n)$ and $O(nlog(n))$ respectively and distance calculations are necessary to build the tree. The disadvantages of the above structure lie in the search operation, where for each node, two distance operations are applied, which results in a higher cost of the search, especially the chosen pivots no guarantee of the optimal partition of the space, making the degeneration of the indexes.

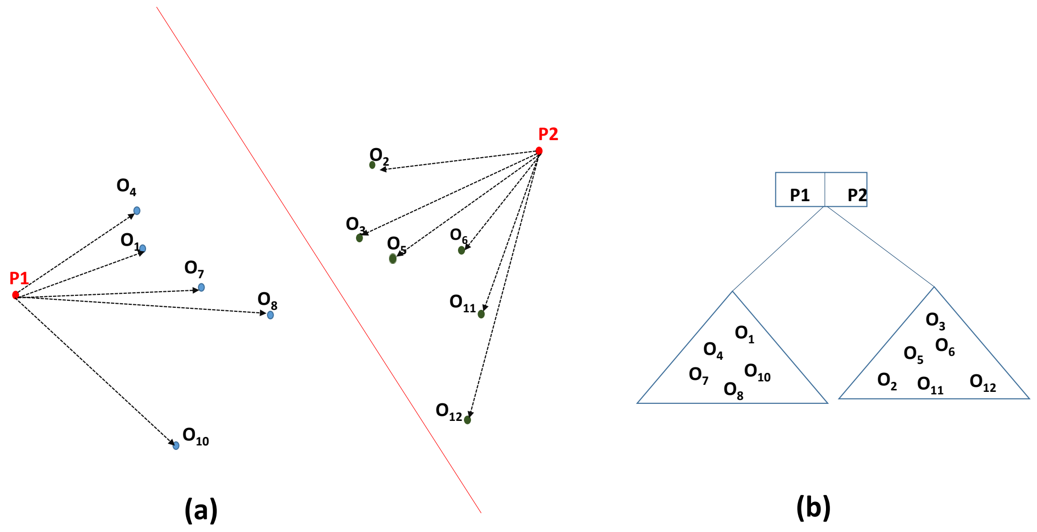


Figure 4.7: (a) Hyperplane space partitioning (b) Structure of the GH-tree [221].

- Geometric Near neighbor Access Tree (GNAT) is a static and m-ary tree [222]. It is a generalization of the GH-tree. The difference is that GNAT uses n pivots in each internal node instead of two pivots (Figure 4.8). The data set $\mathcal{O}$ is divided recursively into n subspaces $S = \{S_1, S_2 \cdots S_n\}$ by the set of pivots $P = \{p_1, p_2 \cdots p_n\}$. The rest of the objects are assigned to the subspaces according to the closest distance to a pivot of $p_i \in P$. This index increases memory requirements and computational costs due to selecting new sets of pivots repeatedly.

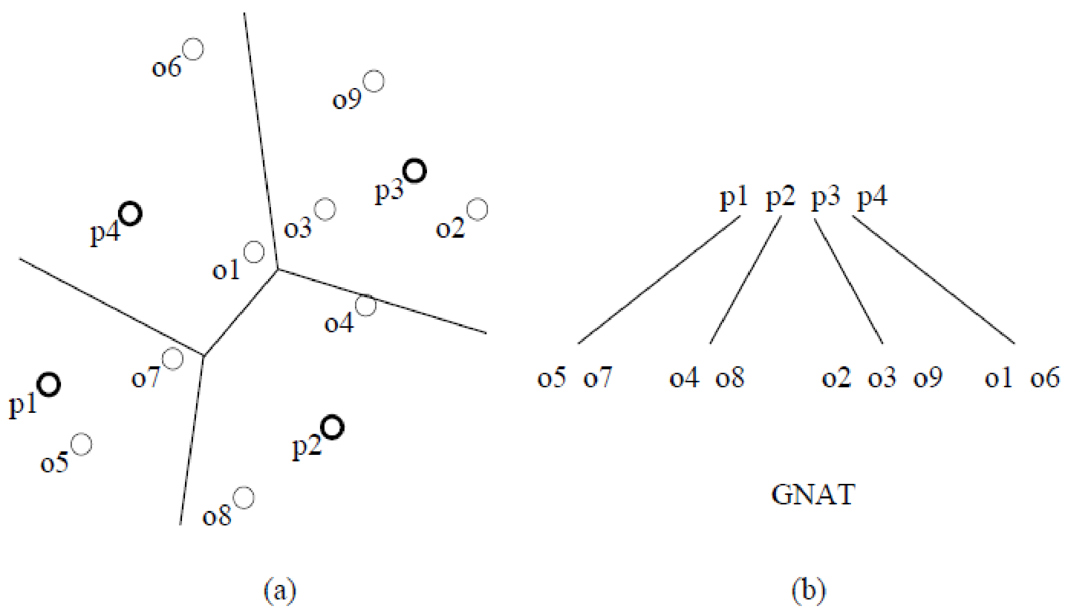


Figure 4.8: Example of partitioning used in GNAT-tree (a) and the corresponding tree in (b) [15].

- Evolutionary Geometric Near-neighbor Access Tree (EGNAT) [223] is an improvement of GNAT. It is dynamic and it allows node organization and update after initial bulk loading, using placeholders for deleted (or changed) objects. However, it does not guarantee the non-overlaps between the inner nodes.

- Balanced Metric space (BM-index) [224] is proposed as a solution for the unbalanced partitions in Voronoi diagram. It is based on pivot permutations scheme in the weighted Voronoi partitioning to eliminate under- and over-filled buckets (Figure 4.9). However, the calculation costs are very high when calculating the time for the convergence of the algorithm of construction and the complexity of the weight of the pivot.

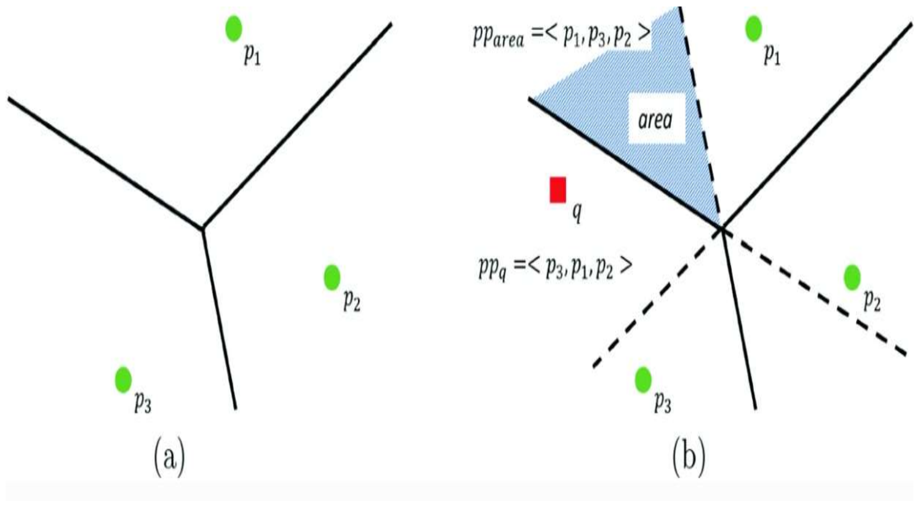


Figure 4.9: Voronoi cells for pivots p_1,p_2,p_3: (a) 1-level tessellation, (b) pivot permutations [224].

- Voronoi Diagram tree (VD-tree) [225] is a dynamic Metric Access Method (MAM). It gathered the coverage radius strategy by the Slim-Tree [226] node partitioning heuristic flexibility and the rigid space partitioning of Voronoi diagrams. VD-tree reduces overlap between nodes by dynamically exchanging overlapped elements. This method eliminates the overlap between nodes. It is based on the displacement of the element furthest from the representative which implies the reduction of the calculation cost. However, the displacement of these elements leads to obtaining unbalanced partitions.

- Complet Hyper-plane tree (CGH-tree) [227] is the combination of GH-tree and mVP-tree [228] by two pivots. It partitions the space recursecement with two hyperbolas and two ellipses. It is not guaranteed the balancing.

Ball partitioning The benefit of ball partitioning is the fact that it only needs one pivot p and the resulting subsets contain the same amount of data, assuming that the median distance d_m is utilized. Several indexes based on ball partitioning have been proposed such as:

- Vantage Point tree (VP-tree) is a binary tree built on the partitioning of the space by the balls as a function of the distance [229]. In VP-tree, the pivot p is selected randomly (Figure 4.10). The median d_m of the distances of the set of objects to the pivot is calculated. Then the median d_m is used to define a ball $B(p, d_m)$ that will divide the space into two disjoint regions. The VP-tree is very costly in terms of distance and time computed, particularly in the high-dimensional data space in which the the number of branches retrieved is great [230].

- Multiple Vantage Points tree (mVP-tree) [231] is a generalization of the VP-tree [229]. It represents an m-ary version of the VP-tree. The nodes are partitioned into several " segments " by concentric rings with the center and equal cardinals instead of one.

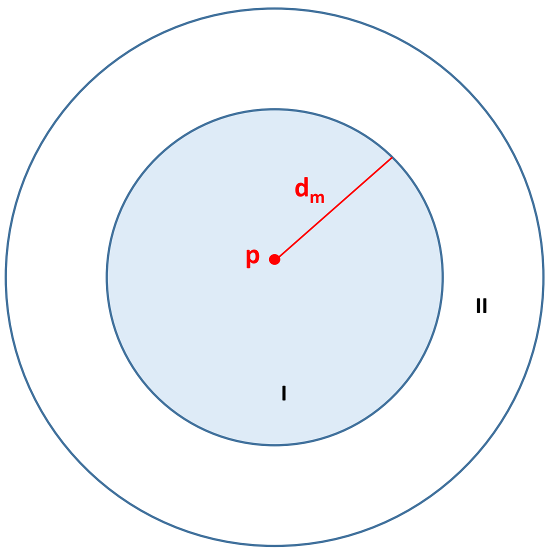


Figure 4.10: Description of the VP-tree.

In fact, it is based on the quantiles rather than the median. The function of this type of index is therefore very similar to that of the VP-tree. The construction time is in $O(n.logn)$. mVP-tree improves the VP-tree [228] and a greater improvement of mVP-tree is obtained by employing many pivots per node [17].

- Memory based Metric tree (MM-tree) [232] this method divides the metric space successively into four regions using two balls which are constructed by two random pivots p_1 and p_2 (Figure 4.11). Region I is the intersection of the balls. Regions II and III are the differences of each ball from the other. Region IV is the rest of space. The distance between (p_1, p_2) is the radius of the two balls . However, the partitioning of the MM-tree may generate subspaces of very different sizes, which involves the production of strongly unbalanced structures [233]. In addition, this index does not support high dimensions.

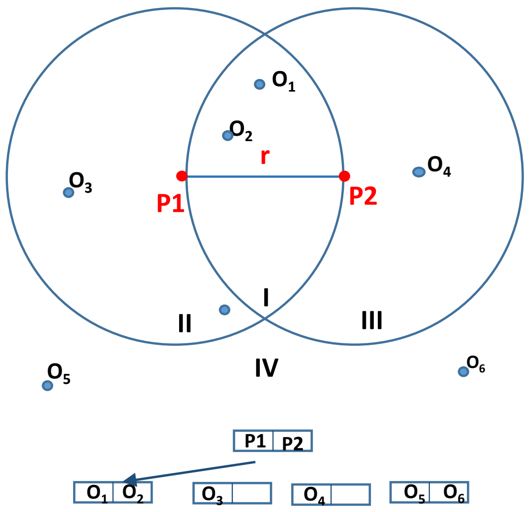


Figure 4.11: Example of a MM-tree indexing of 8 objects [232].

- Onion-tree is the extended version of MM-tree [233]. It recursively partitions space into non-overlapping regions using hyper-spheres to define disjoint subspaces (Figure 4.12). It introduces three features: a partitioning method that controls the number of disjoint subspaces generated at each node, a replacement technique that can change the pivots of leaf nodes in insertion operations and extended query algorithms kNN to support the new partitioning method and including a new visiting order of subspaces. The increase in the number of partitions in the space provides a fast indexing of complex data and accelerates search to answer similarity search queries. However, the issue with the above structure is the extended construction due to the reinsertion of objects.

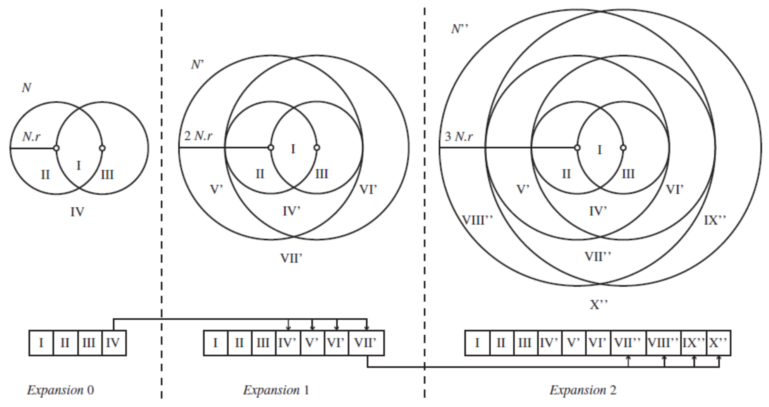


Figure 4.12: Example of two expansion procedures applied to a node N [233].

- Intersection Metric tree (IM-tree) [221] divides recursively the dataset into five disjoint regions, by selecting two farthest points as pivots p_1 and p_2. The fourth region is partitioned into two regions using a plane. Figure 4.13 represents the IM-tree building at a given stage of the recursive splitting process of the dataset. The regions I, II, III, IV and V collapse to level 2. The IM-tree structure is composed of:
  - Leaf nodes: consists a subset of the indexed objects.
  - Internal node:
    - $N1$ for the intersection.
    - $N2$ for the partial ball centred on p_1.
    - $N3$ for the second partial ball centred on p_2.
    - $N4$ for the remaining space close to p_1.

∗ $N5$ for the remaining space close to p_2.

Despite the fact that improves the MM-tree and onion-tree structure, the external region of the balls in the IM-tree, causes the degeneracy of the index for the massive data.

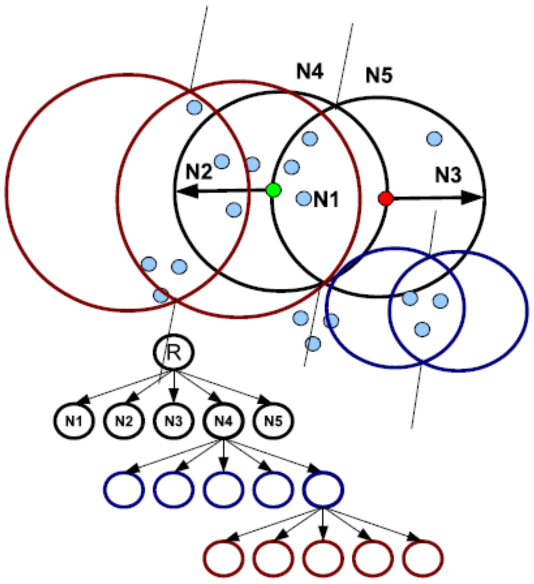


Figure 4.13: Description of the IM-tree [221].

- eXtended Metric tree (XM-tree) [234] divides the space with spheres. They use two structures, sequential and tree structure, to reduce the volume of the outer regions of the spheres, creating extended regions as inspired from the X-tree [177] and inserting them into linked lists as extended regions, and excluding empty sets that do not include any objects. Figure 4.14 represents the XM-tree building at a given stage of the recursive splitting process of the dataset. The regions I, II and III collapse to level 2, the nodes eXt1, eXt2 collapse to the same level. The elements are distributed according to the partitions to which they belong. XM-tree nodes have the following structure: Leaf nodes (objects), internal node (Normal Directory) and extended nodes (chained list of objects). The advantage of this structure is that it is simple

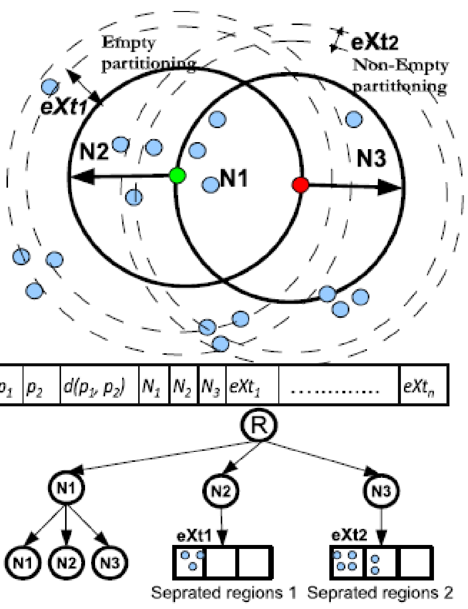


Figure 4.14: Description of the XM-tree [234].

to identify to which partition an element belongs, while ensuring no overlap between nodes of the same level of the tree. In addition the extended regions help to speed up the kNN search due to the exclusion of some objects that are not needed to compute the relative distances of a query object.

- Non-Overlapping Balls and Hyper-planes tree (NOBH-tree) [235] is based on the recursive division of the space into six regions by using two pivots $(p_1, p_2) \in \mathcal{O}$ (Figure 4.15). The rest of the objects are separated so that the evaluation of the distance of an element s_i into p_1 and p_2 can only contain the region S_i. This excludes overlapping regions when answering a point request. The distance between p_1 and p_2 is called the node hop and the regions are divided using both a metric hyperplane and two ball regions, where the radius of the ball r is the node hop. This method suffers from the high cost of insertion and search.

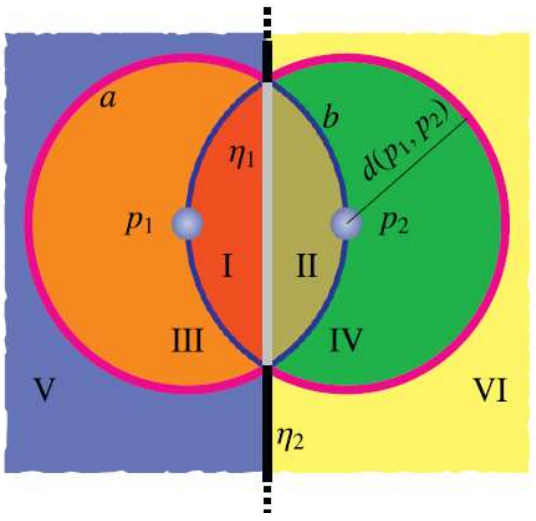


Figure 4.15: Six regions that can be combined to create NOBH-tree members [235].

Mapping pivots partitioning

- D-index is a metric structure at several levels by using the ρ-split functions, one for each level, to create buckets for storing objects [236]. Here, the ρ-split functions of individual levels use the same ρ. In figure 4.16, a ρ-split function based O_7 is used at level 1, and a ρ-split function based on O_3 is used at level 2. Objects in the exclusion bucket '-' (i.e.,O_3, O_5, O_6) at level 1 are candidates to be divided at level 2, and the exclusion bucket of the last level forms the exclusion bucket of the D-index [237].
- eD-index is an extension of the D-index with a modified split function [238].
- iDistance is a B$^+$-tree based dimensional indexing method for similarity search in vector spaces [239]. The iDistance partitions dataset into n clusters C and establishes a reference point p_i for each cluster C_i, $i \in \{0 \cdots n-1\}$. Every object $o \in \mathcal{O}$ is then assigned a numeric key according to the distance

from its cluster's reference object, the iDistance key for an object is:

$$iDistance = d(p_i, o) + i.C \tag{4.1}$$

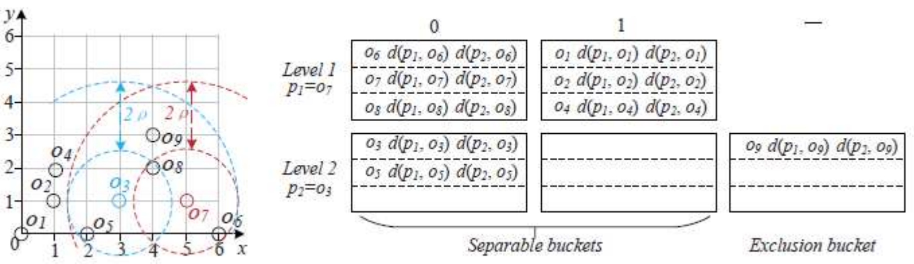


Figure 4.16: Example of the D-index [237].

These reference points are used to transform the space into unidimensional for each partition. The formula maps all objects in any cluster C_i to interval is: [i.C,(i+1).C] (Figure 4.17-a). Mapped objects are indexed by a B^+-tree and the search is performed by one-dimensional range queries. However in a range query $R(q, r)$ several intervals of the iDistance keys determined which need to be accessed in order to process the query (Figure 4.17-b).

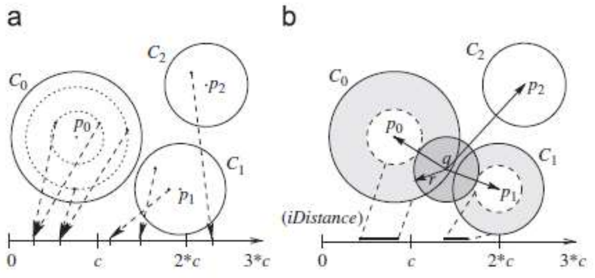


Figure 4.17: Principles of the iDistance [240].

- Metric index (M-index) This index is an extension of the iDistance [240]. It partitions data using Voronoi diagram in several levels (Figure 4.18). In M-index the clusters of iDistances [239] are replaced by the cells of Voronoi. The Voronoi cell centers and the corresponding objects to each cell are mapped by the iDistance method. Mapping of elements of a metric space into a numerical domain, allows the execution of precise and approximate searches algorithms using interval queries.

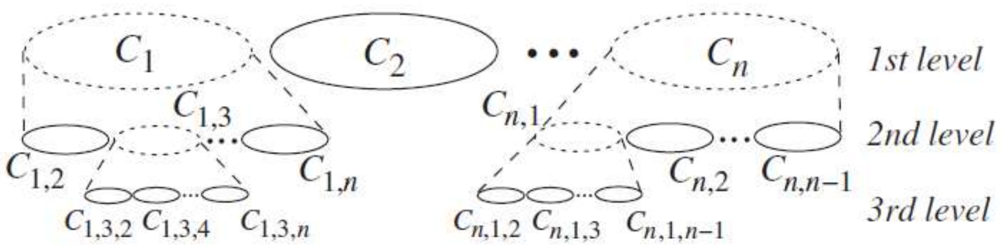


Figure 4.18: Dynamic cluster tree with 3 levels [240].

- Space-filling curve and Pivot-based B$^+$-tree (SPB-tree) [241] In this structure, the SFC (Space Filling Curve) function is used to portion the space into a compact region in the form H and transform the space into a one-dimensional space. The objects mapped by this function are indexed by the B$^+$-tree (Figure 4.19) . The SPBs are components of B$^+$-tree with MBB (minimum bounding boxes)and the objects are stored in a RAF access page. The RAF page stores the objects in the ascending order of their SFC values and build and manipulates B$^+$-tree with minimal cost and minimal storage by regrouping data in compact regions. In addition, it features allow efficient algorithms for handling similarity search and similarity joins. However, the use of parallelism during the construction steps such as space transformations and prepossessing can be done difficultly [152].

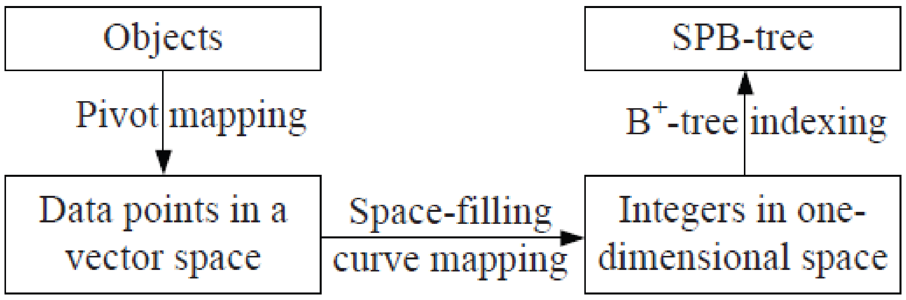

Objects
Pivot mapping
Data points in a vector space
Space-filling curve mapping
Integers in one-dimensional space
B+-tree indexing
SPB-tree

Figure 4.19: Construction framework of an SPB-tree [227].

In D-index [236], eD-index [238], iDistance [239], M-index [240] and SPB-tree [241] for the mapping of the pivots in the space, the data in the metric space are forced with coordinates. However, this mapping is generally deformed. This means that the distance between two points in the metric space generally not equal to the distance between two points in the mapped space.

4.3.1.2 Data partitioning

The data partitioning of the set of points is done by their functions in relation to the selected pivots. Among these techniques we cite the following:

- M-tree is a kind of dynamic and balanced metric trees, which supports consecutive insertion [242]. Its leaf nodes store all the elements, while its internal nodes store selected elements called representatives. Each one of the representatives has a covering radii in which, the data is partitioned into a ball with a pivot and radius (Figure 4.20). This method is dynamic and balanced however, its performance degrades by the overlap between nodes which increases the possibility of multi-way traverse. It is not scalable for high volumes of data [243].

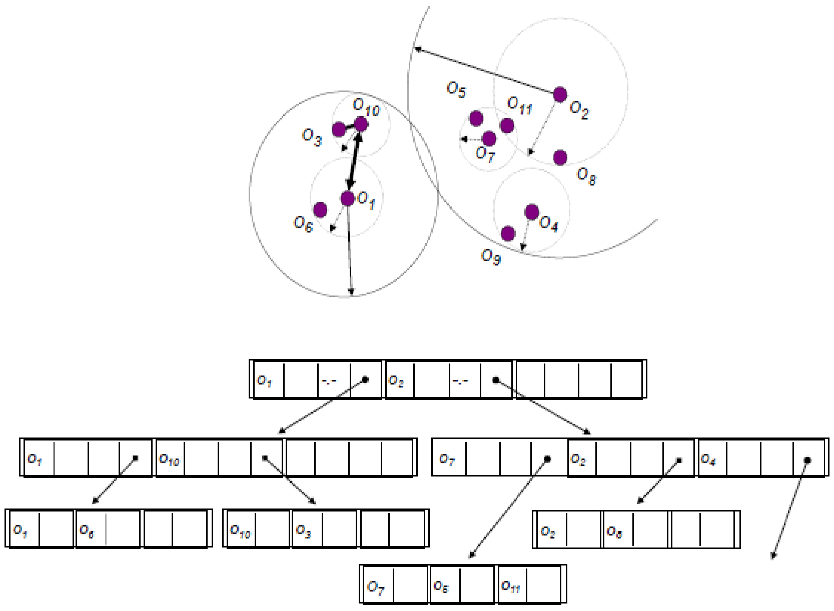


Figure 4.20: Descriptive scheme of the M-tree [18].

- Slim-tree is an improvement of the M-tree [242] in order to reduce the degree of overlap between nodes [226]. It introduces a new splitting technique based on the Minimum Spanning Tree (MST). The main drawback of this structure is the possibility of creating nodes that contain empty nodes, thus strongly limiting the performance of the index, mainly in the case of high dimensional spaces [244].

M-tree [242] and Slim-tree[226] are height-balanced structures that achieve very good performance both in terms of disk access and run time mainly because of the height of the trees are very short. However, the performance of these two structures degrades very easily because the overlap radius of nodes and the overlap between nodes increases such that a large number of subtrees must be analyzed when processing a query [245].

- Density Based Metric tree (DBM-tree) is an extension of the Slim-tree [245]. It was the first dynamic MAM to control overlap, which minimizes the overlap among high-density nodes by relaxing the height balancing rule (Figure 4.21). Subtrees are made deeper in denser regions of the metric space, and less deep in regions with many more objects. It was discovered that reducing the overlap of nodes indeed reduces the number of accesses to the nodes, improving performance. It is a balanced structure. It decreases the overlap between balls and the number of distance calculations when searching a query. Despite these advantages the reorganization of the data for balancing the tree adds an additional computation time.
- DBM*-tree is an improvement of DBM-tree which aims to avoid the recalculation of distances in the choice of the appropriate sub tree during the construction process [246]. The authors proposed a matrix of distances between all objects in each node. However, this solution has the disadvantage that the storage space for the distance matrix at each node is not sufficient in big data.

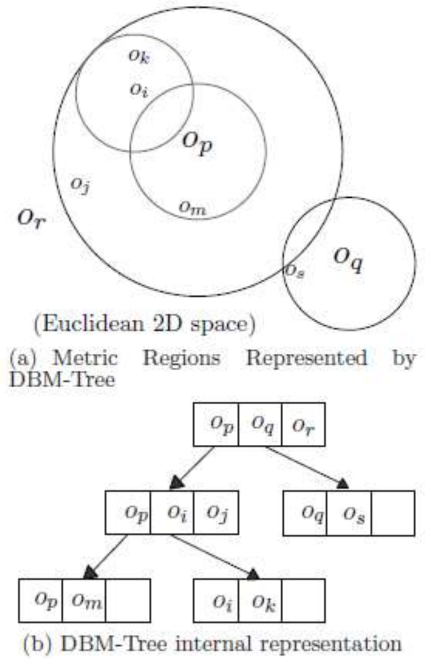


Figure 4.21: Description of the DBM-tree [246].

- M^X-tree [247] is an M-tree extension with the creation of a super node, which is inspired by the X-tree (Figure 4.22). The M^X-tree provides a large search area by extending the super node to the metric spaces completely. A new division method of nodes is introduced in the M^X-tree to address the necessity of the low cost of index construction. In addition, an inner index is proposed in the M^X-tree to transparently manage the CPU costs in the extended leaf nodes due to the introduction of the super node.

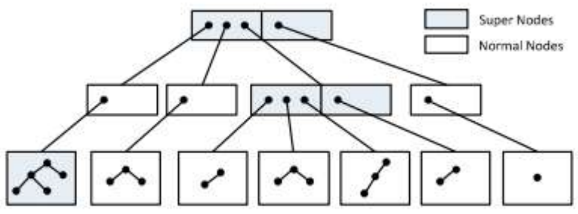


Figure 4.22: Structure of the M^X-tree [247].

- PM-tree in this index, for every leaf overflow, a split algorithm is used to create a new node and to distribute the elements between them [248]. Each node promotes one element to the upper level that stores it and the coverage radius. The upper levels may be updated recursively, if necessary. This process guarantees the structure is always balanced. However, the problem is if an inner node split when splitting an inner node, selecting an element to be promoted and remove it from the node is not possible, as each element is a pivot that represents a branch. The algorithm employs the aggregate nearest neighbor query to solve this problem. This algorithm minimizes the sum of distances to the set of ball pivots, among other aggregation functions. This strategy building compact indexes that increase the performance of k-nearest neighbors. This is achieved due to the faster convergence of the query algorithms.
- Super M-Tree is an extension of the M-Tree [242] where the approximate sub sequence and subset queries become nearest neighbor queries [249]. The authors introduced the spaces of metric subsets as a generalized concept of metric spaces. That use different function the distance, to calculate the distance between objects in internal nodes and its sub tree.
- Hollow-tree is a strategy capable of handling missing data, caused due to the fact that they have not been observed or recorded [250]. It is mainly based on two different techniques CFMLI (Complete First and Missing Last Insert) and ObAD (Observed Attribute Distance). The CFMLI technique is used to index the observed data, with missing values (with NULLS) at the nodes of the leaves while the ObAD technique is applied to compute the set of distance functions, based on the possible combinations of observed and missing values, to estimate the similarity score according to the observed

attributes in the two elements. However, the existence of NULLS values could impact the indexing in a general way due to the fact that the leaves are full but contain NULLS data.

Pre-computed distances methods Other methods use a distance matrix for storing pre-computed distances from each database object to a set of pivots. Among these methods we cite the following:

- Approximating and Eliminating Search Algorithm (AESA) is generally regarded in the literature as the more efficient MAM [251] [252]. It is based on an n^2n matrix of distances between the n objects, which means that it is very costly in terms of calculating the distance to $O(n^2)$, which is why this method is not practical in the case of large data sets. From the authors point of view [251], the AESA is only suitable for for small datasets of at most a few thousand objects.

- Linear AESA (LAESA) is proposed in ordor to minimize the construction cost of AESA [253]. It just needs $O(kn)$ distances in the construction cost. However, it wastes some search efficiency compared to the original AESA.

  AESA and LAESA are static methods. They build an index structure based on fixed data sets and there is no way to insert or remove objects from the structure [254].

- Extreme Pivoting (EP) this index is based on the selection of a set of essential pivots (without redundancy) covering the entire database [255].

- Improvable LAESA (I-LAESA) is an improvement of LAESA [256]. It reduces the calculation of distances in the search for a query. LAESA only uses an exact distance from k pivots to other objects, on the other hand I-LAESA takes an additional estimated value distance. The estimated distance is not expensive to calculate and it is possible to be updated during the search and to be approximated to the exact distance piecemeal.

All these methods use a distance matrix to remove the objects and avoid some distance calculations during the search. Nevertheless, these methods require more space to store the pre-computed distances, and their I/O costs are often high because the data is not clustered in this way [252] [257]. In addition, these methods are not adequate for big IoT data.

4.3.2 Distributed metric space indexing methods

The distributed methods, cited in what follows, are grouped in figure 4.23.

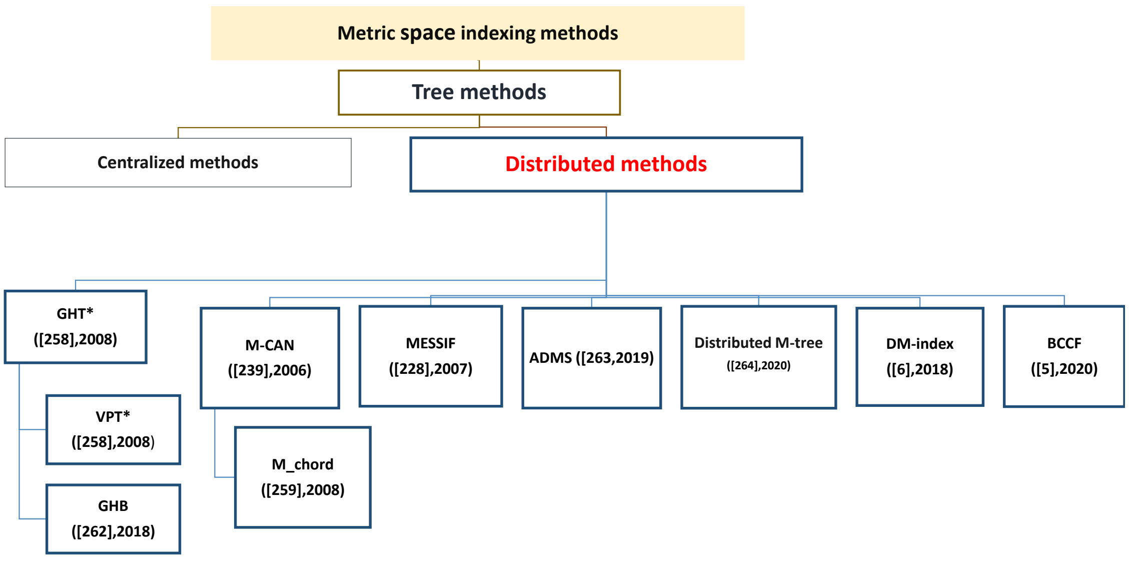


Figure 4.23: Distributed metric space indexing methods.

GHT* Authors in [258] presented the GH-tree in parallel. The aim is the distribution of data storage on several servers. The AST (Address Search Tree) represents a binary search tree that is based on the GH-tree [220]. The GH-tree is generated in each server and client, this tree is charged to the storage and to find the queries. The leaves of this tree are pointers to the buckets BID (Bucket IDentifier) or point to another server by NNID (Network Node IDentifier). This solution is efficient for data distribution. However, they are used for range queries, so they do not cover the kNN search.

VPT* is a VP-tree distributed in a P2P network [258]. The AST (Address Search Tree) represents a binary search tree that is based on the VP-tree [229].This tree is charged to the storage and to find the queries. It is semilar to GHT* [243] in the structure of the inner node and the leaves, with the exception of in the case of the VP-tree structure, just half of the distances are store with respect to GH-tree, as only one pivot is contained in each each inner node.

M-Chord this index uses the M-tree to index local peer data [259]. It is based on the mapping of the data space into a one-dimensional domain and traverses this domain using the Chord routing protocol [260]. The M-chord operates a vector index method iDistance [239] that divides the data space into clusters C_i, finds the reference points p_i in the clusters and defines the one-dimensional mapping of data objects based on their distances from the cluster reference point. When searching a range query, the space to be searched is specified by iDistance intervals for such clusters that intersect the query sphere.

M-CAN it combines CAN and iDistance [239] for similarity search in metric space .In this method, the set of pivots P=$\{p_1, p_2, \cdots, p_N\}$ are used to map objects $o \in \mathcal{O}$ to an N-dimensional vector space $\mathbb{R}^{\mathbb{N}}$ [261]. The used mapping function F(o), applied on the set of objects $\mathcal{O}$, $F : \mathcal{O} \rightarrow \mathbb{R}^N$ is defined as:

$$F(o) = (d(o, p_1), d(o, p_2), \cdots, d(o, p_N)) \tag{4.2}$$

The pivot based filtering is used to reduce the number of evaluated distances. For routing, each peer manages a coordinate based routing table containing the network identifiers and coordinates of its neighboring peers in the virtual $\mathbb{R}^{\mathbb{N}}$ space. In the range query search the routing algorithm transmits the query to the neighbor with the region closest to the target point in the vector space.

The above cited approaches GHT*, VPT* and M-can do not mention algorithms for kNN queries. M-chord and M-can are efficient for data distribution. However, GHT*, VPT* and M-can are used for range queries only, so they do not cover the kNN search.

GHB-tree is inspired from GH-tree [262]. The first idea is to limit the volume of the space. The goal is to eliminate some objects without the need to compute their relative distances to a query object. They proposed a parallel search algorithm on a set of real machine, in p2p network [262]. This tree has two pivots p_1 and p_2 to split the space into left and right regions using a hyperplane (Figure 4.24). The leaf nodes, in the left and right subtrees, contain a subset of the indexed data with a maximum cardinal equal to c_{max}. This index has proven its efficiency during kNN search when compared with the onion-tree and the slim-tree [262].

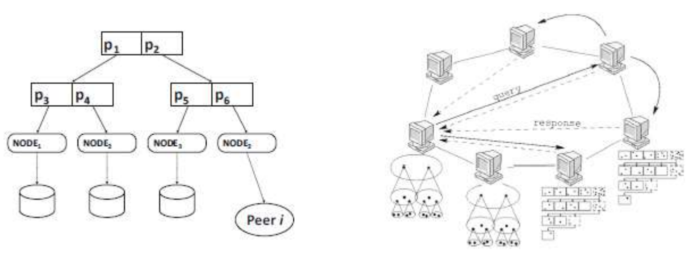


Figure 4.24: Parallel version of GHB-tree [262].

Asynchronous Metric Distributed System (ADMS) is a three-levels distributed architecture for processing similarity queries in large-scale metric spaces

[263]. In the first level (Figure 4.25), the root peer mapped the data to a vector space and requests a mission to distribute the data between the master peers. In the second level the master peers received a mission of data distribution, they communicate with each other to divide the data (using the principle of requester/editor). For the distribution of data they are used Minimum Bounding Box (MBB) of the R-tree . After dividing the data, each master peer is assigned its data set. The data set is divided into equal parts and distributed to their peer workers. In the third level, each peer worker builds its index using M-tree.

In ADMS the objects are recursively divided into disjoint equal-sized partitions, by master peer. They continue to divide their own object fragments into equal size fragments and distribute them to their child peers. The M-Tree is used for Indexing the objects distributed to the peer worker in the vector mapped space. .In additional, they introduced the publish/subscribe communication model to asynchronously exchange messages to decrease time wasting in network interactions.

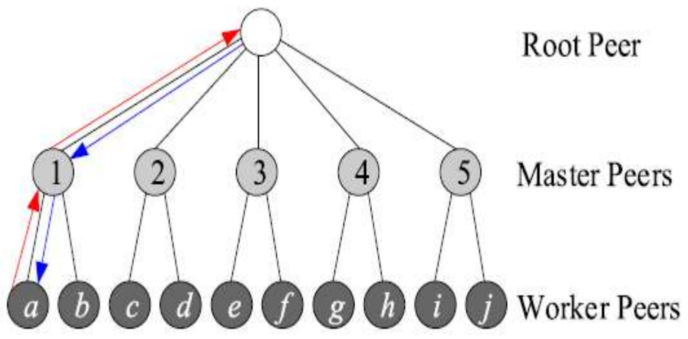


Figure 4.25: Structure of AMDS architecture [263].

Distributed M-tree this method uses M-tree to solve the similarity queries on complex data in multimedia databases only [264]. It is distributed on the Apache Spark framework. The similarity search query uses the kNN algorithm and only the first k response vectors are retained to be sent to the master. The rest of response vectors is ignored. This drawback reduces the efficiency of the kNN search algorithm.

Deployment Model (DM-index) this index was proposed for maintenance and recovery in the fog [6]. It is developed for eliminating redundancy, narrowing the search space and reducing the number of traversed services and recovery time. However, this model index is used only for industrial IoT data and was not tested for other types of IoT data.

Binary tree based on containers at the cloud-fog computing level(BCCF-tree) In this model [5], indexing of IoT data is performed at the fog layers due to their processing power, latency reduction and node distribution. This model can be adapted to emerging IoT technologies to improve the quality of indexing IoT data in real time. In theBinary tree based on containers at the cloud-fog computing level (BCCF), the space is recursively partitioned into two subspaces, centred by two pivots p_1 and p_2 determined using the k-means algorithm with k=2 [104], to achieve a balanced partitioning with minimum overlap in order to reduce the computational cost and the complexity of the similarity search process (Figure 4.26). Despite the efficiency of the BCCF-tree, it presents some inconvenient. The k-means algorithm, which is used during the BCCF-tree construction for overlap decreasing, increased the computational cost and the complexity. In the BCCF-tree, as well as all the metric space methods, indexes degenerate due to the continuous growth of the collected IoT data.

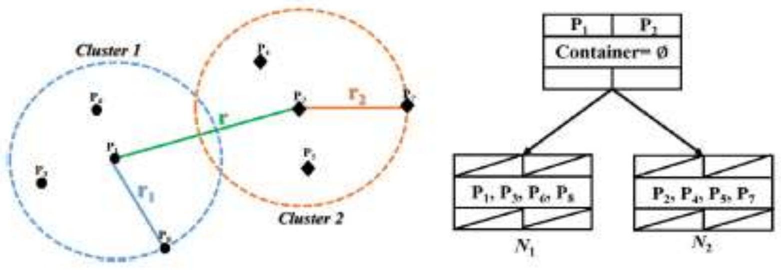


Figure 4.26: Partitioning of space with BCCF-tree [5].

4.4 Comparative Analysis of Indexing Methods

4.4.1 Multidimensional space indexing methods

4.4.1.1 Hashing methods

It is a more useful method in the field of multidimensional data indexing because of its ability to transform a data into a low dimensional element representation (short code composed of a few bits) [160]. The hashing methods are regrouped into Locality Sensitive Hashing methods (LSH) and Learning to Hash methods (L2H). Each method is categorized into centralized or distributed method.

Table 4.1 presents a summary of advantages and disadvantages of LSH methods. In centralized LSH methods several hash tables are necessary to guarantee the quality of the search. However, due to the limitation of storage space and processing capacity of the server, centralized indexing schemes become impractical for big data . Consequently, several distributed indexing schemes based on peer-to-peer (p2p) networks are proposed while how to ensure load balancing remains one of the key issues. In addition to the question of quality guarantee, indexes have to be built for different radii. Thus, in this case, hundreds or thousands of hash tables will be built, resulting in high space and high cost search.

Table 4.1: Summary of locality sensitive hashing methods in multidimensional space.

Category					Advantages	Disadvantages
Multidimensional space indexing methods	Hashing methods	Locality Sensitive Hashing methods	Centralized methods	C2LSH [143]	-Reduce the need to have multiple hash tables and hence reduce the overall index size [265].	-The accuracy of C2LSH was still not high [144]. -It is not scalable.
				QALSH [144]	-Increase the precision of C2LSH by adopting a query-sensitive bucket partition strategy. .	-The guarantee of accuracy during the creation of new projections is an expensive operation. -It suffers from high latency [266].
				PDA-LSH [13]	-It can offer efficient support for both searches and updates.	-The construction of the LMS-tree is very expensive in terms of compaction.
				PM-LSH [142]	-Using the PM-tree improves query processing time. -It uses an adjustable confidence interval to better use distance estimation and provide more accurate results [265].	-It presents the difficulty of estimating the original distance between the points after obtaining the results of a query. -The high storage space consumed by the hash tables. -The complexity of the query search needs the computation of the hash function of a query in addition to the computation of probability on the candidate points.
			Distributed methods	Near bucket-LSH [147]	-It limits the searching process to the compartments to which the query is mapped.	-Insufficient account for the load balancing distribution p2p network.
				LFFIR [148]	-Scalable content-based image retrieval	-Insufficient account for the load balancing problem, that is one of the key issues on the overall performance of the distributed system. [150]
				DSLM [149]	-It can achieve high retrieval rates and mobility resilience.	-Insufficient account for the load balancing problem, that is one of the key issues on the overall performance of the distributed system .
				Balanced and distributed LSH [150]	-It is a balanced distributed indexing scheme.	-It is a static distributed indexing scheme.

Table 4.2 presented a summary of advantages and disadvantages of learning to hash methods (L2H). ODMVH and RDSH are unsupervised methods and the learned hash codes will suffer from limited semantics and discriminative capability. Further, they adopt simple fixed modality weights and binary projection mechanisms, which cannot adapt the variations of streaming multimedia contents and handle the modality-missing problems. In the context of the IoT environment, distributed indexes are very difficult to find the labeling of all different IoT data. Furthermore, they are not suitable for a large and dynamic database and the learning costs are very high.

Table 4.2: Summary of learning to hash methods in multidimensional space.

Category					Advantages	Disadvantages
Multidimensional space indexing methods	Hashing methods	L2H methods	Centralized methods	ODMVH [155]	-It can adaptivly increase hash codes according to dynamic changes in the image.	-ODMVH has limited performance because it is unsepervised and has not exploited any discriminitive semantic information [267].
				RDSH [151]	-It generates a very compact hash code.	-It is not appropriate for a large and dynamic database.
			Distributed methods	DISH [156]	-The requests are distributed in a balanced way.	-High cost time.
				SupDISH [157]	-Effective compact hashing Less memory consumption and calculation cost.	-Difficulty in learning the binary codes.

LSH methods operate with the predefined hash functions regardless of the underlying dataset, where L2H learns custom hash functions based on the dataset. While there is an additional training step necessary, some studies have shown experimentally that L2H outperforms LSH in terms of query search efficiency [158], [159],[160]. However, the Hamming distance is a gross indicator of the similarity between the query and the elements of a bucket as it is discrete and has a limited number of values. Hamming ranking (HR) may not define a good order for buckets having the same Hamming distance from the query. As a consequence, HR generally probes a large unfavorable number of adverse buckets leading to low efficiency. A solution is to employ a long code so that the Hamming distance can classify buckets into larger categories. However, the long code has challenges such as sorting time consumption, high storage demand and low scalability in particular for large-scale datasets [161].

4.4.1.2 Tree methods

Centralized methods Table 4.3 presents a summary of advantages and disadvantages of centralized tree based indexing methods in multidimensional space. These methods are simple and easy to maintain, However, due to the limitation of single-machine resources, they can not support the data generated by the devices of IoT that require high concurrent access to big data, and they are distributed in different regions. In addition, due the considerable increase of volume of data generated by IoT devices, all centralized methods suffer from a common drawback, namely, the degradation of the efficiency of large-scale indexing structures. They are used for the indexing of a specific kind of data and constructed for a predefined dimension of data.

All of the discussed drawbacks show that central indexing in the multidimensional space is not capable of indexing a huge and a growing volume of IoT data.

Distributed methods Table 4.4 presents a summary of advantages and disadvantages of distributed tree based indexing methods in multidimensional space. In distributed indexing methods, because of the data storage architecture, data management models and data processing methods are very different from the centralized system. The indexing structure cannot be easily transplanted into the distributed system. The distributed indexing methods suffer from the location of the data index, the method of accessing the data index and the method of retrieving the data after indexing. Despite the existence of efficient indexing methods in high dimension, each distributed method is used for the indexing of a specific type of data. For example, the S^2R-tree [204] is constructed to index, only, spatial-temporal data and the DAPR-tree [205] is built for, only,indexing geographical coordinates data.

Table 4.3: Summary of centralized tree indexing methods in multidimensional space

Category					Advantages	Disadvantages
Multidimensional space indexing methods	Tree methods	**Centralized methods**	Space partitioning	Kd-tree [162]	-Balanced hierarchical split.	-Costly and arbitrary -Performance limited by data dimension
				KdB-tree [166]	-Balanced structure - Efficient search for point queries.	-Insufficient search performance
				Quad-tree [168]	-Efficient storage and retrieval	-It is not balanced
			Data partitioning	R-tree [12]	-Dynamic and balanced structure.	-It suffers from the problem of overlap between the rectangles. -The time and the complexity of the computation increase in high dimension.
				R*-tree [174]	-Eliminates the overlapping rectangles. -More efficient than the R-tree.	-High complexity.
				R*-tree [175]	-Eliminates the overlapping rectangles. -More efficient than the R-tree.	-High complexity.
				X-tree [177]	-Reduce overlap rate.	-It is very complex.
				SR-tree [172]	-Reduce overlap rate.	-Very costly in insertion and research.
				SUSHI-tree [173]	-Reduce the dimensionality of the feature space.	-The quality of clusters is not guaranteed.
				TPR-tree [178]	-It supports the queries for present and future positions of moving objects.	-It is very difficult to manage R-tree with the change of the number of moving objects and the interval queries cover the whole tree.
				TPR*-tree [179]	-It minimizes the bounding rectangle for reducing query cost[180].	-It cannot handle historical queries.
				D-tree [180]	-Efficiently answers a wide range of queries.	-Not appropriate for the growing number of moving objects.
				BB-tree [176]	-Balanced and dynamic structure.	-Does not support the kNN search.
				B-tree [167]	-Balanced in insertion and deletion. -Efficient for kNN and range search.	-Large storage space is necessary. -Maintenance is costly
				B+-tree [182]	-Storage is minimized.	-High complexity.
				STCB-tree [183]	-Efficient use of storage space. -Reduce index maintenance.	-It is not scalable to support a very high rate of updates.
				UB-tree [181]	-Efficient processing for spatiotemporal data.	-Not suitable for large data.

Table 4.4: Summary of distributed tree indexing methods in multidimensional space.

Category				Advantages	Disadvantages
Multidimensional space indexing methods	Tree methods	**Distributed methods**	General indexing framework [184]	-Scalable and flexible index.	-It is not well suited for cloud systems [185]
			A-tree [186]	-Scalable indexing scheme. -Capable of handling both point and range queries.	-Requires large number of servers. -Limited performance.
			EMINC [185]	-It provides fast query processing. -Efficient index maintenance	-Overlapping in cubes nodes of R-tree.
			CG-Index [189]	-Efficient update and query performance.	-Supports one-dimensional queries.
			RT-can [191]	-Supports point, range and KNN queries search.	-Not scalable regarding the dimensionality of the data.
			LC-index [194]	-Dynamic and supports the operations of insertions and deletions.	-High cost of storage.
			RT-HCN index [193]	-Efficient in space and query search. -Simple and beautiful topology.	-Overlapping in R-tree nodes during the publication.
			RB-index [195]	-Efficient and scalable. -Supports point, range and KNN queries.	-Overlapping in R-tree nodes during the publication.
			U^2-tree} [197]	-Supports point, range and kNN queries search.	-High cost in the update and the maintenance of the distribution index.
			CR-index [198]	-Compact indexing scheme.	-Able to index data with a unique dimension [199].
			CC Index [200]	-Simple structure	-Requires large storage space. -Not support updating indexes after the table has been built.
			UQE-Index [201]	-Support a high insertion rate -Simultaneously provide an efficient multidimensional query.	-Supports only range query.
			SeaCloudDM [203]	-Able to index continuous IoT data.	-Suffers from the latency problem in the cloud computing.
			Multi-attribute index [202]	-Balanced structure. -It can support range queries efficiently.	-Only considering numerical data which is one-dimensional.
			S^2R-tree [204]	-Integrates both spatial and semantic information in the index.	-The conversion of high-dimensional vectors into a low-dimensional space may lose the origin semantic of data.
			DAPR-tree [205]	-Balanced index. -Efficient index for spatial data retrieval.	-It is not dynamic. -Overload when several requests are received.
			Block grid index [13]	-Supports indexing a large-scale moving objects.	-Requires an optimization in the incremental update of the kNN query search when the objects are moving.
			ITTIS [14]	-Suitable for processing temporal data in real time.	-Extensive search time.
			DPISCAN [207]	-Efficient index for large-scale moving objects.	-No query search method.
			Geospatial data indexing} [209]	-Efficient search for range queries for geospatial data.	-Dimension limited to three.
			-Lossless compression Data [211]	-Enhanced of lossless compression indexing in IoT.	-Temporal alignments and deduplication IoT streaming data not addressed.
			Textual and spatial objects indexing [212]	-Minimization of search space by the use of pruning technique.	Difficult update in IoT environment
			$SSKQR^+$ [213]	-Efficient index to retrieve the most relevant data.	-Cost of construction and maintenance.
			Data lakes approach [215]	-Efficient storage for huge amount of data.	-High maintenance cost.
			Haystack queries [215]	-Scalability in both computing and storage senses.	-Costly data load time.
			Hierarchical multidimensional indexing [216]	-Efficient spatial query processing.	-Limited dimension

4.4.2 Metric space indexing methods

Centralized methods Table 4.5 presents a summary of advantages and disadvantages of centralized tree based indexing methods in metric space. These methods are based on the successive division of the space into subspaces. This kind of methods faces the rapid growth of regions and subspaces due to the continuous growth of data which consequently, leads to the degeneration of the index. Another issue is the overlapping between these subspaces which is not solved efficiently. As the volume of data generated by IoT devices has increased considerably, traditional centralized indexing methods became usefulness due to the limitation of the processing capacity which reduces the overall performance of query-based search.

Distributed methods Table 4.6 presents a summary of advantages and disadvantages of distributed tree based indexing methods in metric space. Distributed methods in metric space are able to index any type of data. The distribution of indexes in several local indexes will allow a big data indexing. Nevertheless, the question remains how to distribute the indexes, and how to retrieve the data in these distributed indexes.

Table 4.5: Summary of centralized tree indexing methods in metric space.

Category					Advantages	Disadvantages
Metric space indexing methods	Centralized methods	Space partitioning methods	Hyperplane partitioning	BS-tree [217]	-Simple partitioning. -Reduce overlap rate.	-Static structure. -High computational costs. -High cost search. -Support only range search.
				MBS-tree [218]	-Reduce the cost of search compared with the BS-tree.	-Static structure. -Support only range search.
				V-tree [219]	-Reduce overlap rate. -Simple implementation	-Static structure. -Unbalanced structure. -Reinsertion objects is largely costly.
				GH-tree [220]	-Reduce overlap rate. -Simple implementation	-Static structure. -High cost search.
				GNAT [222]	-No overlapping.	-Static structure. -High computational costs.
				EGNAT [223]	-Needs lower CPU time than the GNAT tree.	-Difficulty in balancing the index.
				VD-tree [225]	-Dynamic structure. -Reduce overlapping rate.	-Unbalanced structure.
			Ball partitioning	VP-tree [229]	-Simple implementation.	-High cost in terms of computed distance and time.
				mVP-tree [231]	-Reduces research costs-	-Static structure. -Support only range search.
				MM-tree [232]	-No overlapping regions. -Dynamic structure.	-Unbalanced structure.
				Onion-tree [233]	-Improved space partitioning as compared to MM-tree. -Dynamic structure.	-Insertion of objects creates a semi-balance.
				IM-tree [221]	-Efficient when comparing to MM-tree and Slim-tree.	-Index degeneration in large-scale data.
				XM-tree [234]	-Fast kNN search.	-High memory requirements.
				NOBH-tree [235]	-No overlapping of the divided data space	-High cost of insertion and search.
			Mapping pivots partitioning	D-index [236]	-Reduction of distance calculations. -No overlapping of the divided data space.	-The mapped of points deformed the distances.
				eD-index [238]	-Suitable for similarity self join.	-Efficient for small query radii only. -The mapped of points deformed the distances.
				iDistance [239]	-Reduction of distance calculations. -No overlapping of the divided data space.	-kNN search performed using one-dimension range search. -The mapped of points deformed the distances.
				M-index [240]	-Reduction of distance calculations. -No overlapping of the divided data space. -Efficient search in comparison with the iDistance.	-The mapped of points deformed the distances.
				SPB-tree [241]	-Reduce the cost in terms of storage construction and search.	-The mapped of points deformed the distances.
		Data partitioning		M-tree [243]	-Dynamic and balanced structure.	-Not scalable for high volumes of data. -High cost search.
				Slim-tree [226]	-Dynamic structure. -Reduce overlap rate.	-Degradation performance in processing a query .
				DBM-tree [245]	-Dynamic structure. -Reduce overlap rate.	-Expensive construction.
				DBM*-tree [246]	-Reduce the cost of construction. -Dynamic structure.	-Needs a high memory space.
				M^X-tree [247]	-Low cost of index. -Dynamic structure.	-High cost of search.
				PM-tree [248].	-Dynamic structure. -Reduce the distance calculations. -Compact indexes increasing the performance of the similarity search.	-High computational costs of construction.
				Super M-Tree [249]	-Able to address approximate requests for subsequences.	-High computational costs of construction.
				Hollow-tree [250]	-Able of handling missing data.	-Not support high volumes of data.
		Pre-computed distances		AESA [251]	-Simple implementation.	-Static structure. -High computational costs of construction. -High cost of storage. –Not support large data sets.
				LAESA [253]	-Reduce the construction cost compared with EASA.-	-Static structure. –Not support large data sets. -High cost of storage.
				I-LAESA [256]	-Reduce the distance calculations in query search compared with LAESA.	-High cost of storage. –Not support large data sets.

Table 4.6: Summary of distributed tree based indexing methods in metric space.

Category			Advantages	Disadvantages
Metric space indexing methods	**Distributed methods**	GHT* [258]	-Parallelism speed up the search. -No overlapping data distribution between servers.	-Support only range search. -Unstable connections between nodes of p2p network
		VP-tree [258]	-Parallelism speed up the search. -No overlapping data distribution between servers.	-Support only range search. -Unstable connections between nodes of p2p network
		M-Chord [259]	-Effective split of local metric based indexes. -Efficient similarity search.	-kNN and range query search performed using one-dimension. -Unstable connections between nodes of p2p network.
		M-CAN [261]	-Effective split of local metric based indexes	-Similarity query search performed using one-dimension. -Unstable connections between nodes of p2p network. -Support only range search.
		GHB-tree [262]	-Balanced structure. -No overlapping data distribution between nodes of p2p network.	-Difficult to balance the index.
		ADMS [263]	-Balanced distributed system.	-Risk of network saturation during message exchange.
		Distributed M-tree [264]	-Support high volumes of data.	-Inefficient kNN search.
		DM-index [6].	-Efficient index for discovery of services. -Adaptive deployment models for fog nodes.	-Unbalanced index in terms of load of fog layer nodes.
		BCCF-tree [5]	-Efficient kNN search. -Balanced partitioning of the data.	-Costly building process. -Degradation in large scale.

4.5 Conclusion

In this chapter, a review of the literature on big IoT data indexing is presented. A new taxonomy of indexing techniques, in both multidimensional and metric spaces, is proposed basing on their grouping into centralized and distributed methods. For the whole indexing methods, in both multidimensional pace and metric space, a comparative analysis was done by pointing out the advantages and the disadvantages of each index. Indexing methods in metric space present better performance compared with the multidimensional space which was awaited since, in metric space, data objects are defined by distances. In the other hand, the few distributed indexing methods in metric space are more efficient than the centralized indexing methods in the space. In the next part, we will propose some distributed indexing methods developed, in this thesis work, in metric space in order to solve some issues raised previously. The similarity query search performance in these indexes will be tested using the kNN search method.

Part II

Propositions

5 Parallel Construction of B3CF-trees

5.1 Introduction

After studying the state of the art of centralized indexes in the multidimensional space or in the metric space, we find that they are limited due to the fact that they suffer from a common drawback of degradation of efficiency in large scale, which makes these methods inefficient for indexing IoT data. This inefficiency also leads to the need for index distribution for ensuring the rapidity of the query search process. The majority of distributed indexes in multidimensional and metric space stored in the cloud [194], [193], [195], [197], which posed various challenges. High or unpredictable latency due the long distances between users and the cloud. High uplink bandwidth requirements, gateways that do not have the bandwidth capacity to upload certain types of sensors data to the cloud will not be able to use the cloud-based storage and processing approach. No in-network filtering or aggregation. Some applications cover a large geographical space, whereas only an aggregate value of the sensors is actually important. Uninterrupted internet connection required [22]. Indexes in the multidimensional space are more robust due to their strong dependency on type or, more precisely, on their geometric properties. This feature makes these indexes only specific for a certain data type which implies that indexing the various types of IoT data is very difficult [39].

To solve these challenges, we propose to relocate the indexing process from the cloud to the fog nodes in order to bring the data as close as possible to the indexing structure and therefore, considerably reduce network congestion. In addition, each fog node generates its own indexing structure, which not only allows parallelism during the construction of trees, but also parallelism in the similarity query search process through the simultaneous launch of the same query on all fog nodes. In each fog node, a clustering method is used as a pre-indexing process. The use of the density-based spatial clustering of applications with noise (DBSCAN) algorithm allows data to be partitioned into homogeneous groups which will be indexed in parallel. This process promotes the creation of a balanced trees with a minimum degree of overlap between the leaves of each tree. Indeed, DBSCAN is a density-based clustering method that stands out for its ability to automatically create clusters with almost zero inter-class similarity. To ovoid the limitations of indexing structures in multidimensional space we choose the metric space. The metric space approach has been found to be very important in building effective indexes for similarity searching. Our index structure is implemented in the metric space because it seems to be the right compromise since, in this space, only distances between data are used regardless of their types and dimensions. In additional, we used tree indexing structures, that is dynamic structures with data changes. The complexity of the insertion and search in tree structure is logarithmic. This means that the search time is reduced logarithmically depending on the number of indexed objects.

In this chapter, we propose a new system for indexing and retrieving data in an IoT environment. The so called Binary tree based on containers at the cloud-clusters fog computing level (B3CF) allows dealing with the index degradation and network congestion while ensuring minimal kNN search time with optimal results quality by the introduction of clustering using the DBSCAN algorithm as a step before indexing. The clustering process, in its turn, allows the introduction parallelism for the indexing of separated clusters and also in the similarity query search using the kNN search method. The proposed approach will be presented in detail followed by the simulation and results of the indexes construction in terms of the number of calculated distances, the number of calculated comparison, the time of indexes construction and the indexes quality. The parallel kNN similarity query search will be also tested by the number of calculated comparisons, the number of calculated distances, the time of search and the number of the visited leaves. The examination of the performance of this proposed index will be performed by comparison with some existing indexes namely BCCF-tree [5], IWC-tree [5], MX-tree [247] and BB-tree [176],[201].

5.2 Proposed Approach

Similarity search queries, in an IoT environment, is very complex due to the exponential increase in data, which needs to be organized. In this approach, the collected data is grouped into clusters, using a clustering algorithm, before their indexing in parallel. Parallelism is an efficient tool to speed up the index construction time and also the search algorithm. Like the BCCF-tree [5], the system architecture consists of three layers: the IoT sensor layer (or terminal layer), the fog layer, and the cloud layer (Figure 5.1). The terminal layer sends the data generated by the interconnected devices to the fog layer. The fog nodes are close to the terminal devices and have the ability to compute and store the data [268]. In this approach, the fog layer is divided into two levels (Figure 5.1). In the first fog level, the data sent by the terminal layer is collected and aggregated. In the second fog level, the data from each cluster is indexed and trees are constructed. The leaves of the nodes in the constructed tree are stored in the cloud layer.

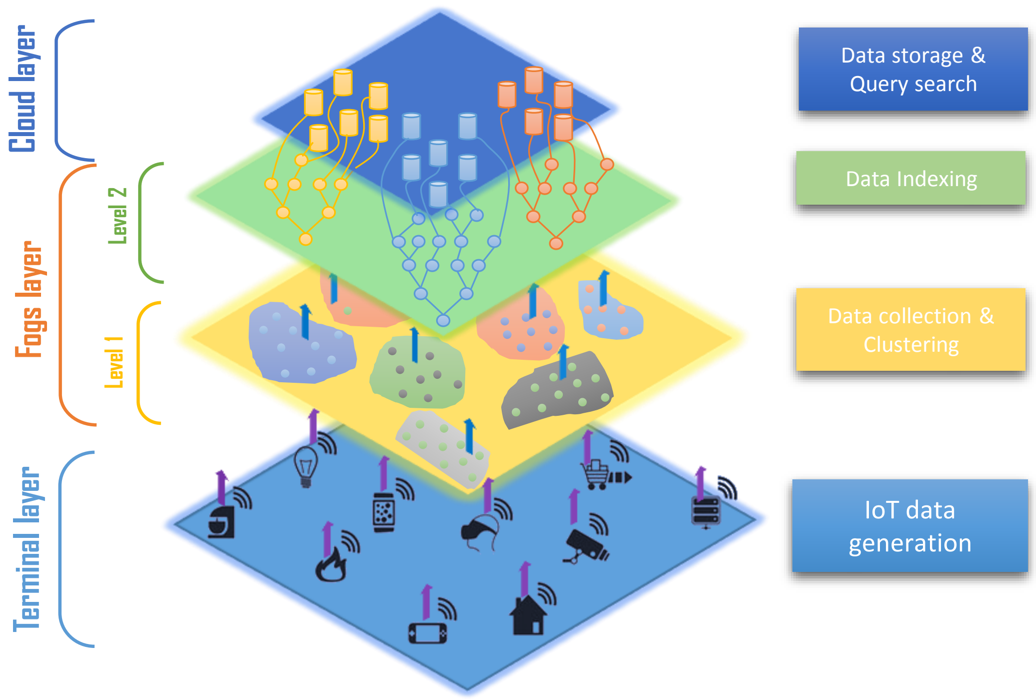


Figure 5.1: Cloud-fog computing architecture.

5.2.1 Clustering fog level

The target of the first fog level is to segment the data gathered from IoT devices. This process can help to construct parallel trees for each cluster and speed up the research since each tree contains only similar objects represented by the root. Indeed, it is not required to go through the entire tree to get a response to the query; in addition, it makes it possible to launch the query on all fog trees at the same time. So to do this, the DBSACN (Density Based Spatial Clustering of Applications with Noise) algorithm (Algorithm 1) was chosen to segment data into clusters. The DBSCAN is a density-based clustering algorithm designed to discover clusters of arbitrary shapes. The main idea of DBSCAN is that, for each object in a cluster, the neighborhood of a given radius must have at least a minimum number of objects.

Table 5.1: Definitions of variables used in algorithms.

Symbols	**Definitions**
O	Set of objects $O = \{o_1, ..., o_{n_s}\}$
n_o	Number of objects in O
C	Set of Clusters $C = \{C_1, ..., C_{n_c}\}$
n_c	Number of cluster
C_c	Center of cluster
$d(a, b)$	Distance function between objects a and b
n_{o_c}	Number of objects in each cluster

In our context, DBSCAN seems to be the better choice since the latter can determine the number of clusters automatically, whereas other clustering methods, such as k-means and spectral clustering, require as input the number of clusters, which is not always easy to determine when dealing with metric and multimodal data.

The basic version of DBSCAN only allows to group similar elements without determining a representative for each cluster. This missing information is very important whether in the phase of construction of the tree or in the phase of finding an element in the tree. Indeed, the representative of a cluster is an optimal choice as the root of the tree knowing that a good root allows optimizing the construction and the research. In addition, the representative makes it possible to reduce the number of comparisons calculated when one wants to select the tree concerned by the search. In practice, the attribution of a class to new data, in the absence of the representant, is carried out by calculating the distance between the new objects and all the elements of the cluster. Creating a representant reduces the number of computed comparisons for each cluster to one. For this, a new version of DBSCAN is proposed (Algorithm 1) to take into account the previously cited requirements. For a better reading of the algorithms, the definitions of all variables are grouped in Table 5.1. In Algorithm 1, clustering of the dataset O, consisting of

n_o objects o, into n_c clusters C with centersC_c is performed using DBSCAN with chosen parameters eps and Minpts.

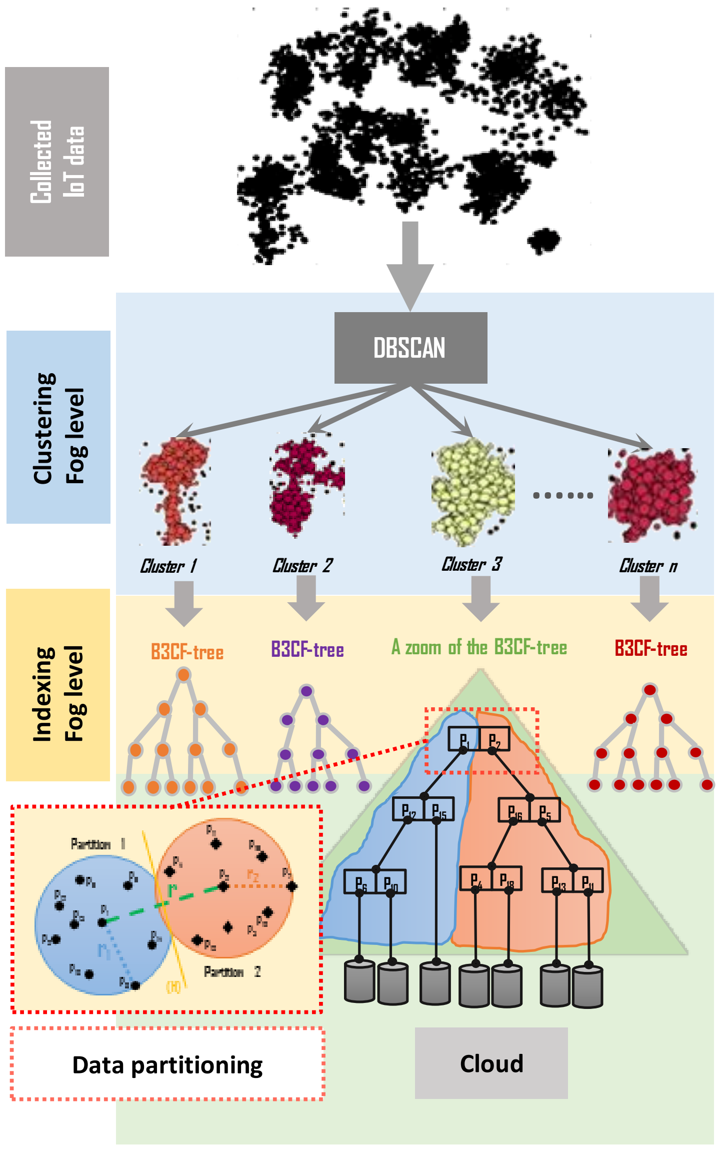


Figure 5.2: B3CF-tree construction in the cloud-fog computing level.

Algorithm 1 DBSCAN modified with cluster centers.

```
Require: $O = \{o_1, ..., o_{n_s}\}$, eps, Minpts
Ensure: $C, Cc$
  ClusterId = nextId(NOISE)
  for $i \in O.size$ do
    Point = o.get(i)
    if $Point.ClId = UNCLASSIFIED$ then
      if ExpandCluster(O , Point, ClusterId, Eps, MinPts) then
        ClusterId = nextId(ClusterId)
      end if
    end if
  end for

  for $i \in \{1..n_c\}$ do
    calcul $Cc_i$
  end for
```

5.2.2 Indexing fog level

After clustering has been done at the first level of the fog, each cluster will be indexed in parallel. The fundamental objective is to allow the construction and interrogation of indexes for clusters of data independently and simultaneously. The aim is to create a dense cluster of objects with small size. To improve the execution time on search algorithms and construction algorithms, compared to our last proposal [5] and also the last existing technique. The aim of the first fog level is to segment the data gathered from IoT devices. This process assists to construct parallel trees for each cluster and speed up the research since each tree contains only similar objects. which limits the volume space, excludes the empty sets; the separable partitions, does not contain objects and creates eXtended regions that will be inserted into a new index. This problem was mentioned by the authors in the field (cruse of dimensionality). The distribution of data has to be almost balanced between all fog nodes. The B3CF-tree (Figure 5.2), a Binary tree based on Containers at the Cloud-Clusters Fog computing level, is strongly inspired by the BCCF-tree [5] and GHB-tree [262] that it tries to improve the performance of the construction and search algorithms of the latter. Space partitioning is a technique that leads to simpler data structures - and thus algorithms. Moreover, the problem of the exponential increase of volumes in large spaces pleads in favor of techniques allowing to reduce or at least to limit the volumes, even to control their occupation, and this is guaranteed by the clustering algorithm DBSCAN. It is based on a partitioning of each cluster, in the metric space, into two regions using two balls at a time.

For the balls construction, we choose two objects and consider them as two pivots (Figure. 5.2). The distance between these two pivots is also the radius of the two balls.

The B3CF-tree nodes - or only N - is defined by:

- L leaf node a set of indexed objects: $E_L \subseteq E$ where $|E_L| \leqslant c_{max}$.
- N Internal node is a septuple: $(p_1, p_2, r, r_1, r_2, N_1, N_2) \in \mathcal{O}^2 \times \mathbb{R}^3 \times \mathcal{N}^2$. where:

  - $r = d(p_1, p_2)$ helps to define two balls B_1 and B_2. According to figure 5.3: $B_1(p_1, r)$ and $B_2(p_2, r)$, centered on p_1 and p_2 respectively and having a common radius value, large enough for the two balls to have a nonempty intersection.
  - r_1 and r_2 are the distances to the farthest object in the subtree rooted at that node N with respect to p_1 and p_2, respectively.
  - N_1 and N_2 are two subtrees(Figure5.3), such that: $N_1 = \{o \in N : d(p_1, o) <= d(p_2, o)\}$ and $N_2 = \{o \in N : d(p_2, o) < d(p_1, o)\}$.

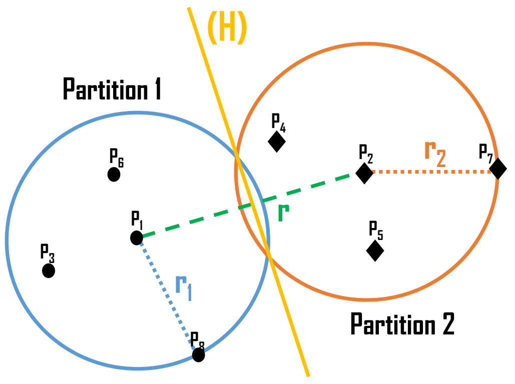


Figure 5.3: Partitioning the space in the B3CF-tree.

5.2.2.1 B3CF-tree build

The construction of the B3CF tree is an incremental process. Algorithm 2 presents a formal description of the parallel index construction process.

Algorithm 2 Parallel B3CF-tree build (C_i, n_{co})

```
Build B3CF-tree $(\in \mathcal{P}()) \in \mathcal{N}$
With:
$(p_1, p_2)=$ The two farthest pivots
```

$$\triangleq \begin{cases} \perp & \text{if S}=\varnothing \\ (e, \perp, \perp, \perp) & \text{if S}=\{\text{e}\} \\ \begin{pmatrix} p_1, p_2 \\ BuildB3CF(\{e \in S : d(p_1, e) \leqslant d(p_2, e)\} \setminus \{p1\}) \\ BuildB3CF(\{e \in S : d(p_2, e) < d(p_1, e)\} \setminus \{p_2\}) \end{pmatrix} & \text{else} \end{cases}$$

The insertion of objects is done from top to bottom (Algorithm 3). Initially, the tree is empty (a leaf encompasses a cluster that contains a set of objects). The farthest two-pivot search algorithm is used for all objects. We have considered putting in place strategies to try to balance the tree, such as choosing two elements furthest apart from each other. After the container will be divided into two non-overlapping subsets so that each element of the container belongs to its nearest pivot. Then, this leaf is replaced by an internal node with p_1 and p_2, and two leaf nodes are created (Figure 5.3).

The data collected at this level of the fog was aggregated into clusters, using the clustering algorithm DBSCAN, before being indexed in parallel with B3CF-tree. The parallelism is an efficient tool to speed up the index construction time as well as the search algorithm.

Algorithm 3 Insertion in B3CF-tree

$$\text{Insert-B3CF-tree}\begin{pmatrix} o \in \mathcal{O}, \\ N \in \mathcal{N}, \\ d \in \mathcal{O} \times \mathcal{O} \to \mathbb{R}^+, \\ c_{max} \in \mathbb{N}^*, \end{pmatrix} \in \mathcal{N}$$

$$\triangleq \begin{cases} (o, \bot, \bot) & \text{if } N = \bot \\ (p_1, o, \bot, \bot) & \text{if } N = (p_1, \bot, \bot, \bot) \\ (p_1, p_2, \text{Insert}(o, d, c_{max}, N_1), N_2) & \text{if } \; N = (p_1, p_2, r, r_1, r_2, N_1, N_2) \\ & \quad \wedge d(p_1, o) \leq r \;\wedge\; d(p_2, o) \leq r \\ (p_1, p_2, N_1, \text{Insert}(o, d, c_{max}, N_2)) & \text{if } \; N = (p_1, p_2, r, r_1, r_2, N_1, N_2) \\ & \quad \wedge d(p_1, o) \leq r \;\wedge\; d(p_2, o) > r \end{cases}$$

The complexity of the B3CF-tree construction is calculated as follows: using the DBSCAN algorithm on the dataset of size n results in clusters of different sizes $n_{oc} < n$. Since the index construction is performed in parallel using Algorithm 2, the complexity can be considered as $O(m. \log m)$ where m is the average size of the resulting clusters. Moreover, the complexity of DBSCAN is $O(n.d)$ where d is the average number of neighbors while the original DBSCAN had $O(n)$ memory complexity [119],[269]. Thus the overall complexity of our approach is $O(n.d) + O(m. \log m)$. Similar to the BCCF-tree [39], the construction follows a balanced hierarchical partitioning of a set of data clusters. The volume of regions becomes smaller, which automatically leads to a lower overlap rate, which in turn will improve the performance of the search algorithms.

5.2.2.2 Parallel kNN seach in B3CF-tree

Similar to the indexing process, parallelism is also used in this work in the similarity search query process to minimize retrieve time. The formal description of the kNN search in the B3CF-tree is summarized in Algorithm 4. The aim of the k-nearest neighbor search is to find the set A of objects closest to a query point q. The kNN search algorithm starts with a query radius r_q initialized to $+\infty$ which should lead to scanning the dataset and then decreases by traversing each tree which corresponds to the distance to the k^e object in the ordered list A. Comparing the distances d_1 and d_2 between the query point q and the two pivotsp_1 and p_2 respectively with r_q indicates the descent of the query point in the index. The leaf nodes contain a subset of the indexed data with a maximum cardinal c_{max}. To find the k nearest neighbors of a leaf, we simply sort the indexed data according to their increasing distances to the query q. As a result of the search, the first k sorted objects are returned.

Because of the parallelisation of the kNN search in all B3CF-trees, the complexity of the kNN search could be reduced to the complexity of search in only one B3CF-tree and it is in the order of $O(\alpha.\sqrt{m}.\log(k)) + \log(m)/(\alpha.k.\sqrt{m})$ where $m = max(n_{oc})$ is the maximum size of the resulting DBSCAN clusters and $\alpha = n_{ovl}/\sqrt{m}$ is the ratio of the number of objects in all visited leaves n_{ovl} to the maximum cardinal $c_{max} = \sqrt{m}$. The first term $O(\alpha.\sqrt{m}.\log(k))$ is the order of the complexity of the computations performed in the leaf while the second term $\log(m)/(\alpha.k.\sqrt{m})$ estimates the complexity of the computations performed when traversing the index from the vertex.

Our proposed approach may have an important impact on IoT data processing due to the closeness of the fog layer from the end user. Any set of heterogeneous IoT data will be able to be indexed using our proposed method because first, it is developed in metric space and second, the use of DBSCAN separates data into clusters of homogenous contents that will be indexed in parallel. The use of parallelism during the data indexing and kNN query search, will speed up both the indexes construction and the similarity search process.

Algorithm 4 Search-kNN in B3CF-tree

$$\text{kNN-B3CF}\left(\begin{array}{l} N \in \mathcal{N}, \\ q \in \mathbb{R}^n, \\ k \in \mathbb{N}^*, \\ d : \mathcal{O} \times \mathcal{O} \to \mathbb{R}^+, \\ r_q \in \mathbb{R}^+ = +\infty, \\ A \in (\mathbb{R}^+ \times \mathcal{O})^{\mathbb{N}} = \emptyset \end{array}\right) \in (\mathbb{R}^+ \times \mathcal{O})^{\mathbb{N}}$$

with :

- $A = ((d_1, o_1), (d_2, o_2), \dots, (d_{k'}, o_{k'}))$;
- $d_1 = d(p_1, q)$;
- $d_2 = d(p_2, q)$;
- $C_1 = B(q, r_q) \cap B(p_1, r) \neq \emptyset$, for the intersection ;
- $C_2 = B(q, r_q) \cap B(p_1, r) \neq \emptyset \wedge B(q, r_q) \cap B(p_2, r) \neq \emptyset$, for the partial ball centered on p_1 ;
- $C_3 = B(q, r_q) \cap B(p_1, r) \neq \emptyset \wedge B(q, r_q) \cap B(p_2, r) \neq \emptyset$, for the partial ball centered on p_2 ;
  - $A_0 = A$;
  - $C_0 = \text{true}$;
  - $r_{q_0} = \min\{r_q, d_{k'}\}$ if $k' = k$ else r_q ;
  - $A_i = \text{kNN-B3CF}(N_i, q, k, r_{q_{i-1}}, A_{i-1})$ if C_i else A_{i-1} ;
  - $r_{q_i} = \min\{r_{q_{i-1}}, d_k\}$ if $|A_{i-1}| = k \wedge A_{i-1} = ((d_1, o_1), \dots, (d_k, o_k))$ else $r_{q_{i-1}}$.

$$\triangleq \begin{cases} A, k\text{-sort}(A \cup \{(d(o,q), o) : o \in L\}) & \text{if } N = L \\ A_2 & \text{if } N = (p_1, p_2, r, N_1, N_2) \end{cases}$$

5.3 Simulation and Results

To test and compare the effectiveness of the proposed approach many experiments were performed on five real data sets with different sizes and dimensions (Table 5.2). The size, or the number of vectors, represents the number of lines in the database while the dimension represents the number of values in each line (Vector coordinates). The databases have been carefully selected from among others to bring together most of the problems encountered in indexing IoT data.

Table 5.2: Characteristics of the selected datasets for the index evaluation.

Dataset	**Size (Vectors)**	**Dimension**
Geographical coordinates	988	2
GPS trajectory	18107	3
Tracking a moving object dataset	62702	20
WARD (Wearable Action Recognition Database)	1000000	5
Smart Home data	5000000	4

1. Geographical coordinate database: a real dataset of 988 2D vectors, which have a low dimensional. It contains BD-L-TC topographic data of selected locations and places [270].

2. GPS trajectory: a dataset of 18107 3D vectors, containing transport trajectories in the northeast of Brazil [271].

3. Tracking of a moving object: a real dataset of 62702 20D vectors. It represents the results of a random simulation of tracking a moving object using wireless cameras .

4. WARD (Wearable Action Recognition Database) [272]: a real dataset of 1000000 5D vectors. It is a reference database for human activity recognition using wearable sensors [273].

5. Smart Home data [274]: a real dataset of 5000000 4D vectors. The dataset

is composed of IoT sensors based on the MQTT communication protocol where the scenario is related to a smart home environment [275].

The experiments were performed using the Python programming language installed on an Intel®CoreTM i7-8550UCPU, 1.80 GHz*8 processor with a 64-bit Linux operating system (Ubuntu). The parameters of the DBSCAN algorithm *Eps* and *Minpts* used for each dataset are regrouped in Table 5.3. In the implementation, the used machine is considered as a fog in which, the received data is processed following two steps.

Table 5.3: Parameter values of the DBSCAN algorithm.

Dataset	**Eps**	**Minpts**
Geographical coordinates	0.062	38
GPS trajectory	70	3
Tracking a moving object dataset	248	250
WARD (Wearable Action Recognition Database)	91	23
Smart home data	170	30

In the first step, for data indexing, two codes were implemented: DBSCAN clustering (algorithm 1) and the parallel build of the B3CF-trees using threads (algorithms 2 and 3). In the second step, the parallel kNN query search is implemented, in threads, using the code of algorithm 4. The effectiveness of the proposed B3CF-tree construction and the query response are tested by comparing our obtained results to those obtained by the following index structures:

- BCCF-tree (Binary tree based on containers at the cloud-fog computing level) [5]: The index B3CF-tree proposed in this study represents an improvement of this index since there is no overlapping of objects when using DBSCAN algorithm for clustering in the metric space.
- IWC-tree (Indexing tree without containers) [5]: The comparison of our

results with those of this index can show the effectiveness of using containers in binary trees.

- MX-tree [247]: The comparison of our results with those of this index can highlight the difference between hyper-plane partitioning and ball partitioning in the metric space.
- BB-tree (Bubble Buckets tree) [176],[201]: This index is constructed in the multidimensional space, a comparison with our proposed index shows the difference between the metric space and the multidimensional space.

5.3.1 Evaluation and comparison of the index construction

The evaluation of the construction index of the B3CF-tree is based on the number of computed distances, the number of comparisons, and the construction time (Figure 5.4) where the size of the containers is set by $c_{max} = \sqrt{n}$. From the obtained results presented in Table 5.4, One can see that the IWC-tree has no results for Smart Home data which reflects the degradation of this index when the data sizes are larger than five million. This is due to the fact that the IWC-tree proceeds with the whole dataset, unlike the B3CF-tree and BCCF-tree which proceed with partial data using means of containers. In the BB-tree index, the balls also act as containers. In the MX-tree, each node has a maximum capacity beyond which it will be divided into two nodes.

5.3.1.1 Number of calculated distances

As shown in Figure 5.4, the number of distances computed during the construction of all index structures changes with the size and dimension of the data sets. The number of distances computed when constructing the proposed B3CF-tree (taken as the sum of number of distances for all clusters) is less than that of the BCCF

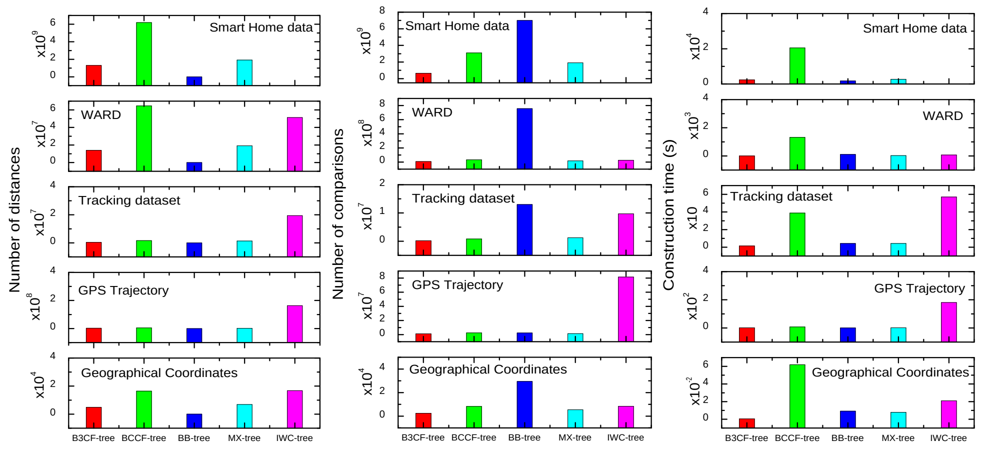


Figure 5.4: Number of distances, number of comparisons and construction time of B3CF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree.

tree, the MX tree, and the IWC tree. In the BCCF-tree, k-means is used for the determination of the two pivots during the index construction and this is what increases the number of distances. In the proposed B3CF-tree, the two pivots are always chosen as the most distant objects. The number of distances calculated during the construction of the B3CF-tree is good compared with other methods except for BB tree because the BB-tree is constructed in the multidimensional space where the data are directly partitioned without calculating distances. The construction of the B3CF-tree is very efficient thanks to DBSCAN algorithm which allows a good data grouping. Indeed, the clusters have the same density and the objects are similar.

5.3.1.2 Number of comparisons

The number of comparisons calculated when constructing the B3CF-tree (also taken as the sum of number of comparisons for the resulting clusters) is lower than that of the other index structures (Figure 5.4) regardless of the space in which they are constructed. This is due to the use of the DBSCAN algorithm for clustering which divides the dataset into clusters of similar (or nearest) objects. For BCCF-tree, the number of comparisons increases due to the use of the whole dataset to build the tree while the BB-tree scored the greatest number of comparisons despite the low number of calculated distances. This is directly related to the construction method which is based on putting the objects to the left or to the right of an axis determined from the medium calculations.

5.3.1.3 Construction time

On Figure 5.4, the construction time of the B3CF-tree (considered as the average time for indexing the resulting cluster data) is less than that of the BCCF-tree and the IWC-tree and close to that of the BB-tree and the MX-tree. For example, the ratio of the build time of the B3CF-tree to that of the BCCF-tree is 0.02% for the geographic coordinate data, 0.8% for the GPS trajectory data, 1.8% for the tracking dataset, 0.47% for the WARD database, and 0.012% for the smart house data. The difference in construction time in the BCCF-tree may be due to the increase in the number of distances, likely related to the use of k-means for pivot determination. For the IWC-tree, since it does not use containers, the distance between objects in all datasets is calculated. Moreover, the parallel construction of indexes from DBSCAN clusters implies an efficient reduction in construction time because the overall size of the dataset is divided over DBSCAN clusters. The indexes construction results confirmed the performance of our proposed approach after comparison with its competitors. Indeed, regrouping the dataset into clusters allows the use of parallelism during the indexing process. In addition, for each B3CF-tree, the choice of pivots as the farthest objects during the partition of data in containers is a simple process but efficient if compared with the k-means method (used in BCCF-tree). The clustering using DBSCAN algorithm before indexing the data , the parallel indexing and the simple manner for the choice of pivots in

Table 5.4: Values of the number of calculated distances, the number of comparisons and the construction time.

	Number of distances					Number of comparisons					Construction time(s)				
	B3CF-tree	**BCCF-tree**	**BB-tree**	**MX-tree**	**IWC-tree**	**B3CF-tree**	**BCCF-tree**	**BB-tree**	**MX-tree**	**IWC-tree**	**B3CF-tree**	**BCCF-tree**	**BB-tree**	**MX-tree**	**IWC-tree**
Geographical coordinates	4.93E+03	1.64E+04	2.00E+01	6.86E+03	1.68E+04	2.46E+03	8.22E+03	2.95E+04	5.46E+03	8.39E+03	4.00E-04	1.66	1.43	3.27	2.33E+03
GPS Trajectory	2.39E+06	4.3E+06	2.60E+01	1.46E+06	1.63E+08	1.19E+06	2.46E+06	2.39E+06	1.43E+06	8.16E+07	6.19E-02	7.65	38.80	1.32E+03	2.05E+04
Tracking Database	3.96E+05	1.64E+06	1.80E+01	1.35E+06	1.94E+07	1.98E+05	8.20E+05	1.30E+07	1.24E+06	9.70E+06	9.20E-03	0.50	4.27	1.06E+02	1.75E+03
WARD	1.40E+07	6.47E+07	2.60E+01	1.91E+07	5.13E+07	7.00E+06	3.23E+07	7.57E+08	1.76E+07	2.56E+07	7.80E-03	1.67	4.20	26.70	2.70E+03
Smart Home data	1.30E+09	6.19E+09	5.38E+02	1.92E+09	–	6.51E+08	3.10E+09	7.01E+09	1.91E+09	–	2.09E-02	1.80E+02	57.10	67.10	–

the containers contributed efficiently in the reduction of the construction cost.

5.3.2 Evaluation and comparison of the constructed index quality

To check the quality of the constructed B3CF tree, the number of nodes per level, the distribution of data in the leaf, the number of internal nodes, the number of leaves, and the tree height features were examined compared to the BCCF-tree, BB-tree, MX-tree and IWC-tree. Table 5.5 lists the values of the last three features.

5.3.2.1 Number of nodes per level

The number of nodes per level varies according to the dataset as shown in Figure 5.5. It is constant for the GPS trajectory and Smart Home datasets and varies from level to level for the other datasets. The number of nodes per level is plotted for three clusters (result of the DBSCAN algorithm) in the Geographic Coordinates and GPS trajectory datasets. However, for the WARD and Smart Home datasets, the DBSCAN algorithm gave more than three clusters and therefore only three clusters were chosen to present the results. According to Figure 5.5, the proposed B3CF-tree index structure is efficient for computing very large data.

5.3.2.2 Data distribution in leaves

Figure 5.6 shows the data distribution in the leaves. The B3CF-trees in each cluster of the Tracking and WARD datasets are balanced. This is because the data for both datasets are well distributed between the left and right sides of each tree. For the Geographic Coordinates dataset, only the index of the first cluster is balanced. Our index is very efficient, and this is because the space is divided

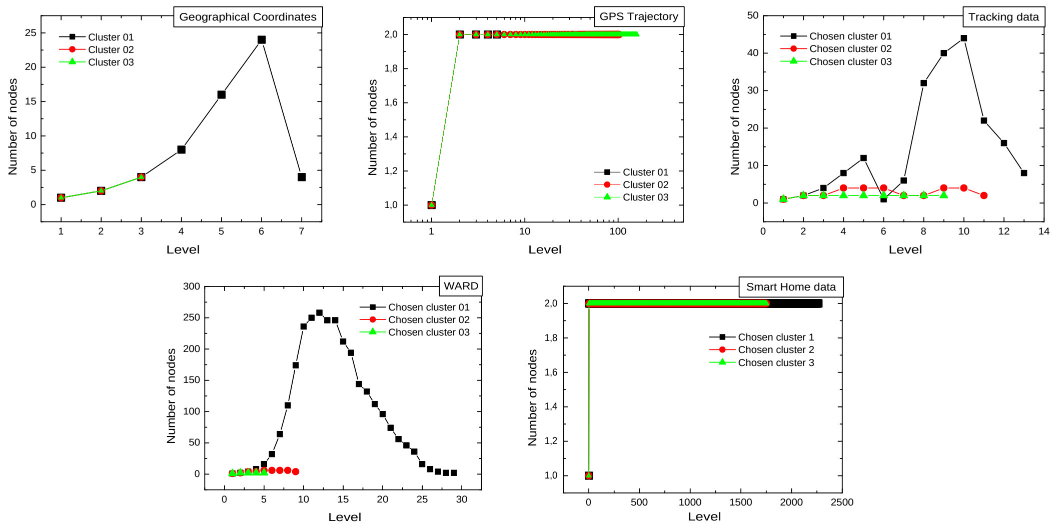


Figure 5.5: Number of nodes per level in the B3CF-tree.

into two sub-parts that do not intersect. This method ensures that the nodes do not overlap. In addition to this, the application of DBSCAN to the first level data makes the data in each group similar and close to each other, which makes the tree composition balanced.

5.3.2.3 Number of internal nodes

The number of internal nodes in the B3CF-tree is lower than that of other index structures (Figure 5.7). The number of internal nodes in the IWC-tree structure is high because it does not use containers that control the partitioning of data.

5.3.2.4 Number of leaf nodes

The same observations can be made for the number of leaf nodes with respect to the relationship between it and the number of internal nodes (Figure 5.7). The number of leaf nodes in the B3CF-tree is lower than in the other structures because the use of the DBSCAN algorithm implies the grouping of the closest objects in

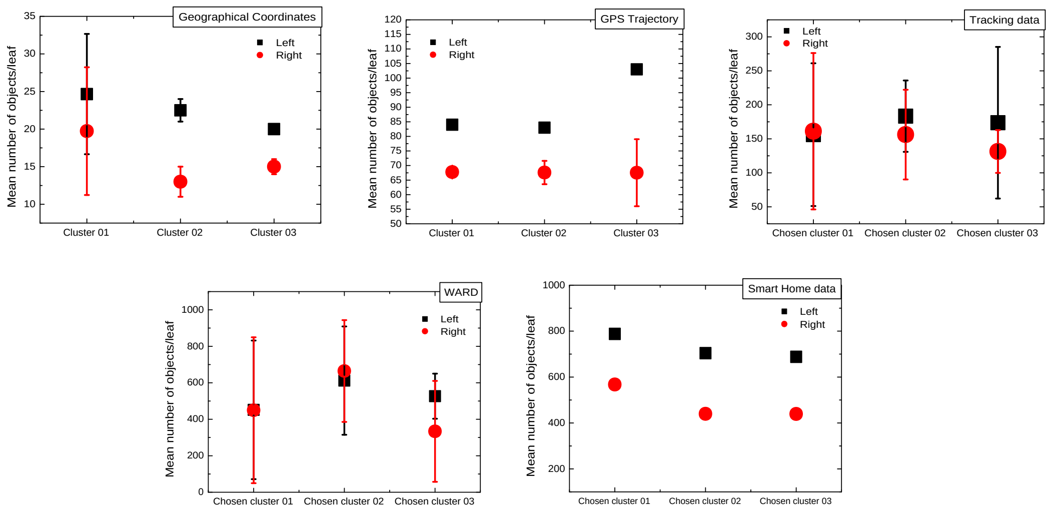


Figure 5.6: Distribution of data in the B3CF-tree.

the same leaf.

5.3.2.5 Tree height

The height of the B3CF-tree varies from one data set to another. It is close to those of the other structures for the geographic coordinate and GPS trajectory datasets and higher than those of the other structures for the other datasets. The high height of the B3CF tree reflects the effectiveness of clustering using the DBSCAN algorithm in partitioning the data. After analyzing and comparing the statistical results of the construction and quality of the B3CF-tree, it can be deduced that the proposed index structure performs well. This is due to the use of a combination of the DBSCAN algorithm in clustering and the parallelism method during the construction of the cluster index. This combination allows a fast construction of the index without overlapping nodes. Indeed, the use of the DBSCAN algorithm guarantees the creation of clusters without overlapping data. On the other hand, when building the B3CF-tree, the choice of the two most distant objects, inside the

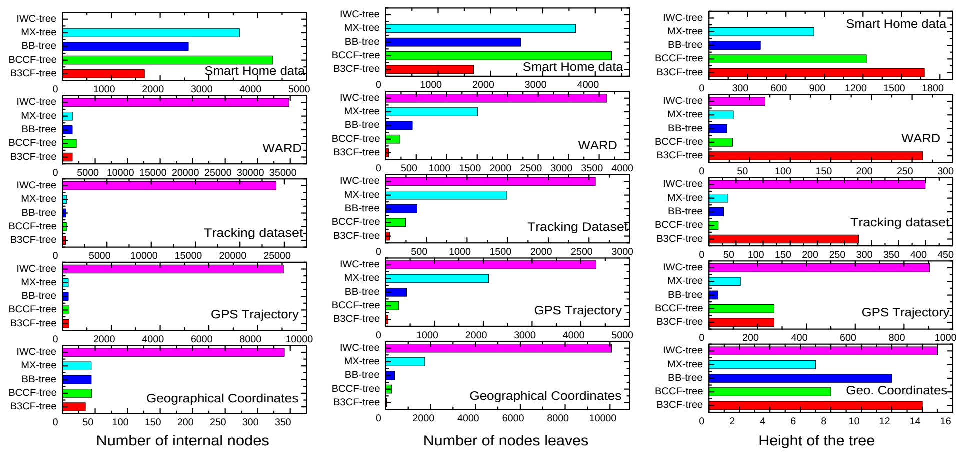


Figure 5.7: Number of internal nodes, number of nodes leaves and height of B3CF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree.

Table 5.5: Values of the number of internal nodes, the number of the nodes leaves and the height of the tree.

	Number of internal nodes					**Number of nodes leaves**					**Height of the tree**				
	B3CF tree	BCCF tree	BB tree	MX tree	IWC tree	B3CF tree	BCCF tree	BB tree	MX tree	IWC tree	B3CF tree	BCCF tree	BB tree	MX tree	IWC tree
Geographical coordinates	35	45	44	44	341	39	46	45	45	288	14	8	12	7	15
GPS Trajectory	264	266	233	232	9052	267	267	243	233	2	267	17	22	30	9052
Tracking Database	339	425	384	434	24077	393	426	385	435	10005	276	29	25	35	399
WARD	1489	2108	1489	1507	348109	1741	2109	1490	1508	220874	263	267	37	129	69
Smart Home data	1667	4311	2577	3626	–	1678	4312	2578	3627	–	1678	1226	44	818	–

containers, as pivots guarantees the partitioning of the space into two parts, which ensures the non-overlapping of the nodes and the good balancing of the index. All these criteria can allow a fast search when searching the similarity query. To test the effectiveness of our B3CF-tree, the results of the kNN search will be presented and discussed in the next section.

5.3.3 Evaluation and comparison of the kNN search

For the evaluation of kNN search with $k = 5, 10, 15, 20, 50$, and 100 in the proposed B3CF-tree index structure, the number of distances, number of comparisons, search time and number of visited leaves will be determined to reach the 100 queries. When examining the search efficiency of similarity queries, the obtained statistical results were compared with those of the BCCF-tree, MX-tree, BB-tree and IWC-tree indexing structures. Note that all statistical results were averaged over 100 randomly generated queries.

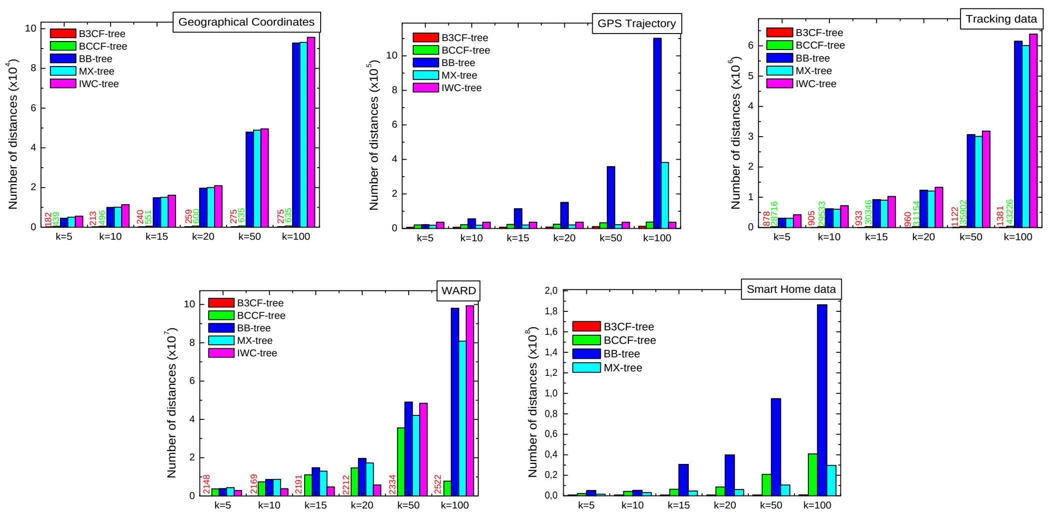


Figure 5.8: Number of calculated distances for the kNN search in B3CF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree.

5.3.3.1 Number of calculated distances

Figure 5.8 shows the number of calculated distances for number of neighbors k between 5 and 100. As can be seen, the proposed B3CF-tree has the smallest number of calculated distances compared to the BCCF-tree, the BB-tree, the MX-tree and the IWC-tree. We can see, also, that the number of distances calculated in the B3CF-tree (Figure 5.8) is nearly invariant as a function of the number of neighbors k between 50 and 100. For all databases used in this evaluation, the ratio between the number of distances of $k = 50$ and $k = 100$ varies between 1.00% and 2.00%. This result reflects the efficiency of the parallel search in our proposed structure. Even the number of calculated distances is high in the BCCF-tree, it is not affected, for some databases, by the increase of the number of neighbors k like the BB-tree, the MX-tree and the IWC-tree. Table 5.6 summarizes the values of the number of calculated distances.

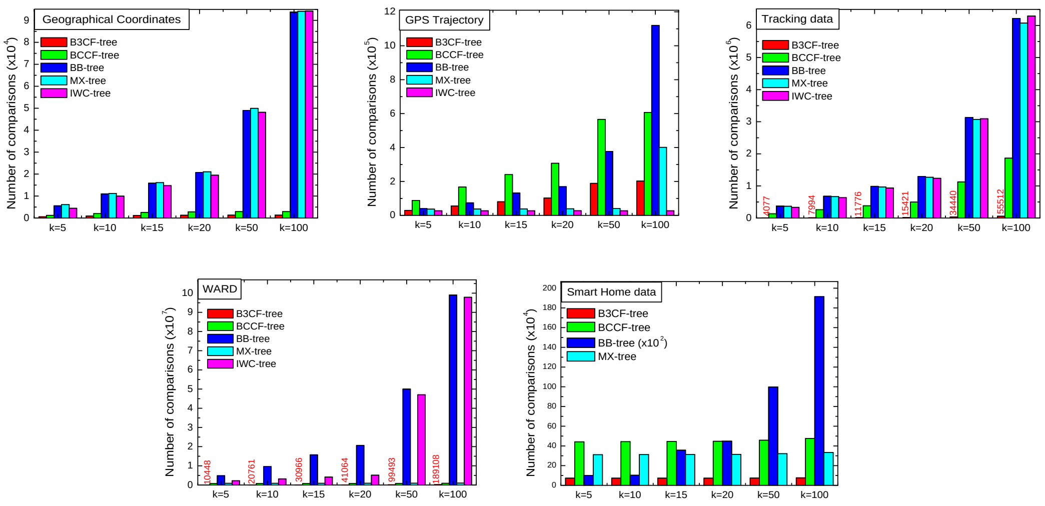


Figure 5.9: Number of comparisons calculated for the kNN search in B3CF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree.

5.3.3.2 Number of calculated comparisons

As can be seen in Figure 5.9, for all the used datasets, the lowest number of comparisons corresponds to the proposed B3CF-tree, except for the GPS trajectory data where the IWC-tree has the lowest number of comparisons and almost constant regardless of the value of k. However, for the same dataset, when we compare the variation of the number of comparisons in the B3CF-tree as a function of the parameter k with those of the other index structures, one can observe that the variation of the number of comparisons in the B3CF tree follows a saturation law which indicates that the number of comparisons stabilizes for a value of k greater than 100. This is not the case for the BB-tree, for example, where the evolution of the number of comparisons follows an exponential law. Even though the number of comparisons in the MX-tree is lower than in our B3CF-tree, it increases about 10 times when $k = 100$. Table 5.7 lists the values of the number of comparisons calculated.

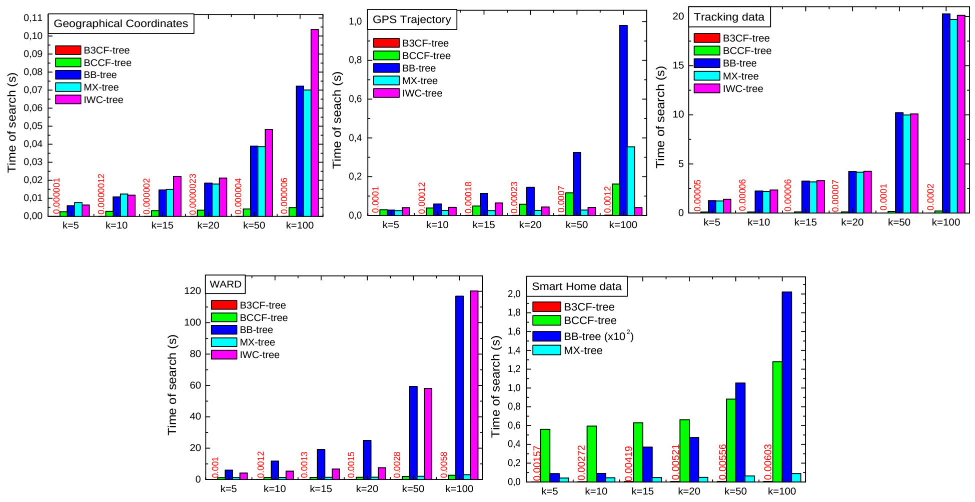


Figure 5.10: Time of kNN search in B3CF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree.

5.3.3.3 Time of search

According to Figure 5.10, the proposed B3CF-tree has the lowest search time compared to the BCCF-tree, the BB-tree, the MX-tree and the ICW-tree structures. The ratio of the search time of the BCCF-tree to the B3CF-tree is 0.07% for the geographical coordinates data, 0.55% for the GPS trajectory data, 0.06% for the tracking dataset, 0.13% for the WARD database and 0.55% for the smart home data. We observe that the value of k has no influence on the performance of the search algorithm. In the B3CF-tree, the ratio of the search time of $k = 50$ and $k = 100$ is 1.56% for the geographical coordinates data, 1.74% for the GPS trajectory data, 1.82% for the tracking dataset, 2.07% for the WARD database and 1.07% for the smart home data. Our proposed index exhibits the shortest search time not only by comparison with the chosen structures, but also, by comparison with other indexes. For example, according to Zhang et al.[185], the combination of the R-tree and KD-tree (EEMINC) answered the point query in 1 thousand nodes and 10 million records in time between 40 and 50 ms and, according to Hu et al. [210], the execution of the 8 nearest neighbours query on the hierarchical index method of 5.5 billion points takes 8.49 s. For smart home data of 5 millions vectors (Figure 5.10), the B3CF-tree answers the average of 100 queries in a time between 0.0016 and 0.006 s. This indicates that the use of clustering coupled with parallelism significantly improves the efficiency of the kNN search by decreasing the search time. The search time values are grouped in Table 5.8.

Table 5.6: Number of distances calculated for the kNN search in B3CF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree.

	Geographical coordinates					GPS Trajectory					Tracking Database					WARD					Smart Home data			
	B3CF-tree	BCCF-tree	BB-tree	MX-tree	IWC-tree	B3CF-tree	BCCF-tree	BB-tree	MX-tree	IWC-tree	B3CF-tree	BCCF-tree	BB-tree	MX-tree	IWC-tree	B3CF-tree	BCCF-tree	BB-tree	MX-tree	IWC-tree	B3CF-tree	BCCF-tree	BB-tree	MX-tree
k=5	1.80E+02	4.40E+02	4.50E+03	5.10E+03	5.60E+03	6.80E+03	2.10E+04	2.20E+04	1.90E+04	3.60E+04	8.80E+02	2.90E+04	3.10E+05	3.00E+05	4.20E+05	2.10E+09	3.70E+12	3.80E+12	4.40E+12	2.80E+12	7.40E+05	2.10E+06	5.10E+06	1.50E+06
k=10	2.10E+02	5.00E+02	1.00E+04	1.00E+04	1.10E+04	7.30E+03	2.20E+04	5.50E+04	1.90E+04	3.60E+04	9.10E+02	3.00E+04	6.20E+05	6.00E+05	7.20E+05	2.20E+09	7.40E+12	8.60E+12	8.70E+12	3.80E+12	7.50E+05	4.30E+06	5.30E+06	3.10E+06
k=15	2.40E+02	5.50E+02	1.50E+04	1.50E+04	1.60E+04	7.70E+03	2,30E+04	1.10E+05	2.00E+04	3.60E+04	9.30E+02	3.00E+04	9.20E+05	9.10E+05	1.00E+06	2.20E+09	1.10E+13	1.50E+13	1.30E+13	4.80E+12	7.60E+05	6.40E+06	3.10E+07	4.60E+06
k=20	2.60E+02	6.00E+02	2.00E+04	2.00E+04	2.10E+04	8.20E+03	2.50E+04	1.50E+05	2.00E+04	3.60E+04	9.60E+02	3.10E+04	1.20E+06	1.20E+06	1.30E+06	2.20E+09	1.50E+13	2.00E+13	1.70E+13	5.70E+12	7.70E+05	8.50E+06	4.00E+07	6.10E+06
k=50	2.80E+02	6.40E+02	4.80E+04	4.90E+04	5.00E+04	1.10E+04	3.30E+04	3.60E+05	2.20E+04	3.60E+04	1.10E+03	3.60E+04	3.10E+06	3.00E+06	3.20E+06	2.30E+09	3.60E+13	4.90E+13	4.20E+13	4.80E+13	8.20E+05	2.10E+07	9.50E+07	1.10E+07
100	2.80E+02	6.40E+02	9.30E+04	9.30E+04	9.60E+04	1.20E+04	3.70E+04	1.10E+06	3.80E+05	3.60E+04	1.4.0E+03	4.30E+04	6.20E+06	6.00E+06	6.40E+06	2.50E+09	7.70E+12	9.80E+13	8.10E+13	9.90E+13	9.00E+05	4.10E+07	1.90E+08	3.00E+07

Table 5.7: Number of comparisons calculated for the kNN search in B3CF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree.

	Geographical coordinates					GPS Trajectory					Tracking Database					WARD					Smart Home data			
	B3CF-tree	BCCF-tree	BB-tree	MX-tree	IWC-tree	B3CF-tree	BCCF-tree	BB-tree	MX-tree	IWC-tree	B3CF-tree	BCCF-tree	BB-tree	MX-tree	IWC-tree	B3CF-tree	BCCF-tree	BB-tree	MX-tree	IWC-tree	B3CF-tree	BCCF-tree	BB-tree	MX-tree
k=5	0.55	1.21	5.55	6.12	4.44	2.90E+04	8.80E+04	4.00E+04	3.70E+04	2.70E+04	4.10E+03	1.30E+05	3.70E+05	3.70E+05	3.30E+05	1.00E+04	7.60E+05	4.80E+06	8.80E+05	2.20E+06	7.30E+04	4.40E+05	1.00E+07	3.10E+05
k=10	0.92	2.02	11.00	11.10	9.98	5.60E+04	1.70E+05	7.40E+04	3.80E+04	2.70E+04	8.00E+03	2.60E+05	6.80E+05	6.70E+05	6.30E+05	2.10E+04	7.70E+05	9.60E+06	8.90E+05	3.20E+06	7.30E+04	4.40E+05	1.00E+07	3.10E+05
k=15	1.16	2.54	1.59	16.10	14.70	8.00E+04	2.40E+05	1.30E+05	3.80E+04	2.70E+04	1.20E+04	3.80E+05	9.90E+05	9.70E+05	9.40E+05	3.10E+04	7.80E+05	1.60E+07	9.00E+05	4.10E+06	7.30E+04	4.50E+05	3.60E+07	3.10E+05
k=20	1.27	2.81	20.70	21.00	19.50	1.00E+05	3.10E+05	1.70E+05	3.90E+04	2.70E+04	1.50E+04	5.00E+05	1.30E+06	1.30E+06	1.20E+06	4.10E+04	7.90E+05	2.10E+07	9.00E+05	5.10E+06	7.30E+04	4.50E+05	4.50E+07	3.10E+05
k=50	1.32	2.91	49.00	49.90	48.20	1.90E+05	5.70E+05	3.80E+05	4.00E+04	2.70E+04	3.40E+04	1.10E+06	3.10E+06	3.10E+06	3.10E+06	9.90E+04	8.30E+05	5.00E+07	9.40E+05	4.70E+07	7.40E+04	4.60E+05	1.00E+08	3.20E+05
k=100	1.32	2.91	93.80	94.10	94.30	2.00E+05	6.10E+05	1.10E+06	4.00E+05	2.70E+04	5.60E+04	1.90E+06	6.20E+06	6.10E+06	6.30E+06	1.90E+05	8.90E+05	9.90E+07	1.00E+06	9.80E+07	7.60E+04	4.80E+05	1.90E+08	3,30E+05

Table 5.8: Time of the kNN search in B3CF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree

	Geographical coordinates					GPS Trajectory					Tracking Database					WARD					Smart Home data			
k	B3CF	BCCF	BB	MX	IWC	B3CF	BCCF	BB	MX	IWC	B3CF	BCCF	BB	MX	IWC	B3CF	BCCF	BB	MX	IWC	B3CF	BCCF	BB	MX
5	7.50E-07	2.00E-03	6.00E-03	8.00E-03	6.00E-03	9.50E-05	2.90E-02	2.70E-02	2.40E-02	4.00E-02	5.30E-05	1.10E-01	1.30E+00	1.20	1.40	9.90E-04	1.20	5.90	1.20	4.10	2.00E-03	5.60E-01	8.80	4.20E-02
10	1.20E-06	3.00E-03	1.10E-02	1.20E-02	1.20E-02	1.20E-04	3.80E-02	5.90E-02	2.50E-02	4.10E-02	6.10E-05	1.10E-01	2.20E+00	2.20	2.30	1.20E-03	1.30	1.20E+01	1.30	5.30	2.70E-03	5.90E-01	9.10	4.40E-02
15	1.70E-06	3.10E-03	1.90E-02	1.50E-02	2.20E-02	1.80E-04	4.90E-02	1.10E-01	2.50E-02	6.50E-02	6.10E-05	1.20E-01	3.20E+00	3.20	3.30	1.30E-03	1.30	1.90E+01	1.40	6.70	4.20E-03	6.30E-01	3.70E+01	4.60E-02
20	2.30E-06	3.40E-03	1.80E-02	1.80E-02	2.10E-02	2.40E-04	5.70E-02	1.40E-01	2.60E-02	4.40E-02	6.90E-05	1.20E-01	4.20E+00	4.10	4.20	1.50E-03	1.40	2.50E+01	1.50	7.50	5.20E-03	6.60E-01	4.70E+01	4.90E-02
50	3.90E-06	4.10E-03	3.90E-02	3.90E-02	4.80E-02	6.90E-04	1.20E-01	3.20E-01	2.80E-02	4.10E-02	1.10E-04	1.60E-01	1.00E+01	10	1.00	2.80E-03	1.90	5.90E+01	2.10	5.80E+01	5.60E-03	8.80E-01	1.10E+02	6.40E-02
100	6.10E-06	4.80E-03	7.20E-02	7.00E-02	1.00E-01	1.20E-03	1.60E-01	9.80E-01	3.50E-01	4.00E-02	2.00E-04	2.10E-01	2.00E+01	20	2.00	5.80E-03	2.70	1.20E+02	3.10	1.20E+02	6.00E-03	1.30E+00	2.00E+02	8,90E-02

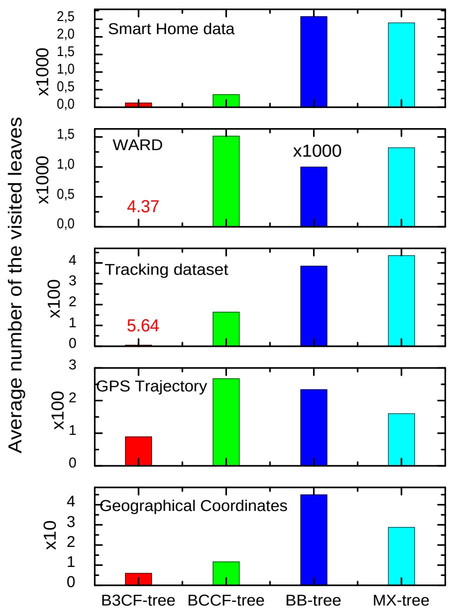


Figure 5.11: Number of the visited leaves in B3CF-tree, BCCF-tree, BB-tree and MX-tree.

5.3.3.4 Number of the visited leave

Figure 5.11 shows the number of leaves visited during the kNN search in the B3CF-tree, BCCF tree, BB tree and MX tree. Note that the number of visited leaves is invariant to the number of neighbors k which is between 5 and 100. The IWC tree is not considered because this structure does not support kNN search [5]. Indeed, this structure presents poor results according to Figure5.8, 5.9 and 5.10. As can be seen in Figure 5.11, the B3CF-tree presents the smallest number of visited leaves and that is why the search time in our index is low. This is due to the use of DBSCAN algorithm for clustering which induced non-overlapping clusters. Figure 5.11 shows the number of the visited leaves during the kNN search in B3CF-tree, BCCF-tree, BB-tree and MX-tree. It is to notice that the number of the visited

Table 5.9: Average number of the visited leaves in B3CF-tree, BCCF-tree, BB-tree and MX-tree.

Number of the visited leaves	Geographical Coordinates	GPS Trajectory	Tracking Database	WARD	Smart Home data
B3CF-tree	6.025	89	5.465	4.36918	120.36
BCCF-tree	11.65	267	163.84	1513.39	359.04
BB-tree	45	234	385	1000000	2578
MX-tree	28.74	160	435	1320.32	2400.25

leaves is invariant as a function of the number of neighbors k which is between 5 and 100. The IWC-tree is not considered because this structure does not support the kNN search [5]. Indeed, this structure presents poor results according to Figure 5.8, 5.9 and 5.10. As can be seen in Figure 5.11, the B3CF-tree exhibits the lowest number of the visited leaves and that is why the time of search in our index is low.This is due to the use of the DBSCAN algorithm for clustering which induced no-overlapping clusters. The number of the visited leaves is regrouped in Table 5.9.

5.4 Conclusion

This chapter presented a new indexing structure called B3CF-tree(Binary tree based on Containers at the Cloud-Clusters Fog computing level) the indexing process is delocalized from the cloud to the fog nodes to get the data near the indexing structure and thus reduce the network traffic congestion significantly. Moreover, each fog node creates its unique indexing structure, allowing not only parallelism in tree construction, but also parallelism in the search process by launching the same query simultaneously on all fog nodes. Second, a post-index process is performed in each fog node. It partitions the data into similar groups using the DBSCAN algorithm. The aim of this process is to generate a balanced tree with a reduced degree of overlapping between the leaves of the tree. Indeed, DBSCAN is a density-based clustering method that is distinguished by its ability to automatically create clusters with almost zero inter-class similarity.

6 CV Method for Indexing Continuous IoT Data

6.1 Introduction

In the previous chapter, we have presented the B3CF-tree that is tested for a unique data stream. IoT data from devices are continuously generated in multi types such as textual, numerical, streaming and multimedia data [199]. Storing this continuous streams of IoT data and finding an efficient retrieving method is a big challenge regarding the dynamicity and the diversity of types and dimensions.

In this chapter, in order to index continuous stream of IoT data and finding an efficient retrieving method. We propose an effective approach, in the fog-cloud computing level, to organize and store continuous IoT data stream and make rapid the similarity query search. Because it is collected from different devices, in the terminal layer, IoT data is characterized by heterogeneity, noise, diversity and rapid growth [85]. For the organization of each IoT data stream, the fog layer is divided into three levels: clustering fog level, clusters processing fog levels and indexing fog level. In the clustering fog level, DBSCAN is used for clustering because it is the most suitable algorithm for grouping diverse IoT data into homogeneous and high density clusters. Each cluster of the first data stream is stored in the clusters processing fog level and directly indexed in a BH-tree (Binary tree with Hyper-plane) in the indexing fog level. For the arrival data streams, after DBSCAN clustering, the indexing is based on the comparison of the coefficient of variation (CV) value of the arrival cluster and those of the union of the arrival cluster with the existing clusters in the clusters processing fog level. According to the minimum value of CV, the arrival cluster is directly indexed in a new BH-tree or, is inserted in an existing index.

The proposed approach will be detailed in what follows. The simulation and results will be presented and discussed by the comparison with two other scenarios .The first scenario is called Creation of a New Index (CNI method) and the second scenario is called Insertion in an Existing Index (IEI method). The comparison in terms of the number of calculated distances, the number of calculated comparisons and the construction time during the trees construction process. The same parameters were tested and compared with two other scenarios in the kNN query search method. The consumed energy during the parallel kNN search is also presented and discussed. For the storage and the indexing of the continuous IoT data stream, we benefit from the cloud-fog computing architecture (Figure 8.1). In the terminal layer. IoT devices, geographically distributed, generate continuously large and diverse data. The indexing of this continuous data stream is proceeded in the fog-computing layer because of its numerous characteristics such as the reduction of the latency of services, the providing of real-time applications and the capacity of processing of high number of nodes [53]

6.2 Proposed Approach

For the storage and the indexing of the continuous IoT data stream, we benefit from the cloud-fog computing architecture (Figure 8.1). In the terminal layer. IoT devices, geographically distributed, generate continuously large and diverse data. The indexing of this continuous data stream is proceeded in the fog-computing layer because of its numerous characteristics such as the reduction of the latency of services, the providing of real-time applications and the capacity of processing of high number of nodes [53].In this work, the fog layer is divided into three levels: the clustering fog level, the clusters processing fog level and the indexing fog level (Figure 6.1). In the clustering fog level, each data stream, from the terminal layer, is grouped into homogenous clusters. Clusters of the first data stream are stored in the clusters processing fog level and their objects are directly indexed in separated BH-trees in the indexing fog layer. For the arrival data streams, according to the Coefficient of Variation (CV) value of their clusters, in the clusters processing fog level, a new BH-tree will be constructed or objects of the arrival cluster will be inserted in an existing BH-tree.

The processing capabilities of fog nodes are not affected by the additional work introduced in each layer since the amount of sensors installed will automatically give rise to a suitable type of hardware to capture, process and transmit data from the sensors. This means that a large number of sensors implies additional power from the fog (This condition is ensured in the installation process). In addition, fog's three-level architecture with specialisation of each level allows smoother processing.

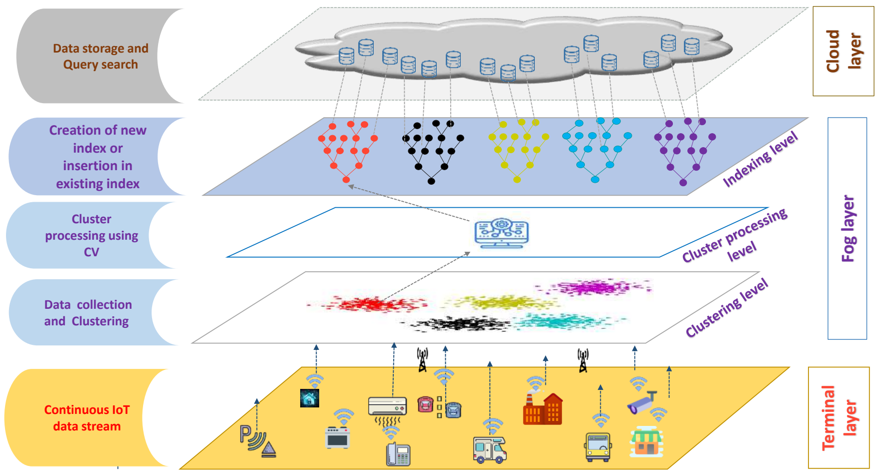

Data storage and Query search
Creation of new index or insertion in existing index
Cluster processing using CV
Data collection and Clustering
Continuous IoT data stream
Cloud layer
Indexing level
Cluster processing level
Clustering level
Fog layer
Terminal layer

Figure 6.1: Architecture of the CV method for indexing continuous IoT data.

In what follows, a detailed description of the clustering, the CV and the indexing methods will be presented. The definitions of the used parameters are regrouped in Table 6.1.

6.2.1 Clustering method

In the clustering fog level, each data stream, sent by the terminal layer, is collected and grouped in N clusters Cl_n with $\{n = 1..N\}$ using the DBSCAN algorithm (Density-Based Spatial Clustering of Applications with Noise) [276] modified by the introduction of the calculation of the clusters centers, noticed in this work c_n, for the coefficient of the variation (CV) calculation. Each cluster Cl_n contains similar elements.

The triggering of the clustering process is closely linked to the storage capacity of the fog node since the fog nodes do not have the same storage and processing capacities. This condition makes it possible to go beyond congestion and conceptual bottleneck and allows tailoring processing with the capabilities of the fog node.

DBSCAN algorithm is one of the most used data clustering method [277]. This algorithm is based on the connection of points within specific distance threshold. However, it connects only those points that satisfied a density threshold (minimum number of objects in a radius). The DBSCAN algorithm partition the data into clusters of arbitrary shapes. Each cluster contains all the objects that are connected by the density. The choice of this clustering method came from the fact the DBSCAN clusters are automatically formed while the k-means algorithm, for example, requires the determination of the number of clusters before clustering. Also, the DBSCAN algorithm is robust in the detection of outliers which are considered as objects that wait for other similar objects in the next data stream. The complexity of the DBSCAN algorithm for grouping a dataset of o objects into N clusters is $O(o.d)$ [278] where $o = oc_1 + oc_2 + \ldots + oc_N$ which could be written as $o = N.mean(oc)$, where oc is the number of objects in per cluster, and d is average number of neighbours. That gives us the final form of the complexity of the DBSCAN algorithm for each data stream which is $O(N.mean(oc).d)$.

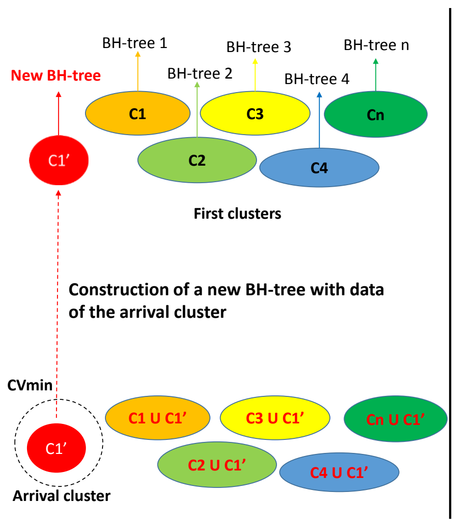
6.2.1.1 CV method

In the clusters processing fog level, the coefficient of variation (CV) is used as a criterion to decide if a cluster of the arrival data stream is to be inserted in an existing BH-tree or indexed in a new BH-tree. The coefficient of variation is a statistical measure of the dispersion of data points in a dataset around the mean. It represents the ratio of the standard deviation to the mean. The advantage of the use of the coefficient of variation is that it is not sensible to the data type and dimension [279]. The clusters processing fog level contains clusters of the first data stream Cl_n. In this fog level (Figure 6.2), each cluster of the arrival data stream Cl'_k is unified with a copy of all the existing clusters Cl_n (Algorithm 5).

Table 6.1: Table of notations.

Abbreviation	Explantation
N	Number of the first clusters
K	Number of the arrival clusters
Cl_n, $\{n = 1..N\}$	Clusters of the first data stream
$Cl_k^{'}$, $\{k = 1..K\}$	Clusters of the arrival data stream
c_n,$\{n = 1..N\}$	Cluster centers of the first data stream
$c_k^{'}$, $\{k = 1..K\}$	Cluster centers of the arrival data stream
$Cl_k^{'} \cup Cl_n$	Union of the arrival clusters $Cl_k^{'}$ and the first clusters Cl_n
$d(c_n, c_k^{'})$	Distance between two centers
I_n, $\{n = 1..N\}$	Set of indexes
Min_d	Minimum distances between the centers of the existing clusters and the incoming clusters
p_1,p_2	Pivots
E	Set of elements
LN	Leaf node
IN	Inner node
o	Object
L	Left sub tree
R	Right sub tree
q	Query
r_q	Radius for recovering k objects closes to q
A	List in with, the set of k objects is stored
$B(q, r_q)$	Query ball q with radius r_q

After that, the CV of the cluster of the arrival data stream $CV_{Cl_k^{'}}$ and the CV of the union of this cluster with every existing cluster $CV_{Cl_k^{'} \cup Cl_n}$ are determined. If the cluster of the arrival data stream $Cl_k^{'}$ has the minimum value of CV, a new BH-tree is constructed, in the indexing fog level, and the cluster $Cl_k^{'}$ is stored with the existing clusters Cl_n in the clusters processing fog level. If the minimum value of CV correspond to the union of the cluster of the arrival data stream with an existing cluster $Cl_k^{'} \cup Cl_n$, objects in the arrival cluster $Cl_k^{'}$ are inserted in the BH-tree of the corresponding existing cluster Cl_n.



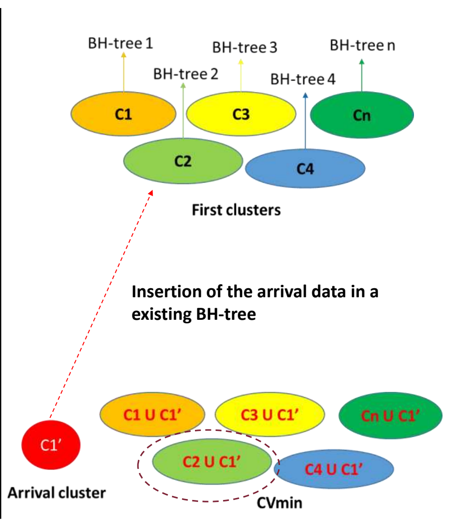


Figure 6.2: CV method in the cluster processing level.

Because the CV calculation of the union of one arrival cluster with the first clusters is parallel, the complexity for all clusters is taken as the complexity for the CV calculation of the cluster with a maximum number of objects oc_{max} which represents approximately $2mean(oc)$ and it is given by $O(mean(oc))$. Due to the fact that the comparison of N arrival clusters with existing clusters is sequential, the complexity of the CV method for each data stream is $O(N.mean(oc))$. The CV method processes the clusters and not the data themselves, this makes it possible to considerably reduce the processing time, despite a polynomial complexity, because the number of clusters is negligible compared to the number of data. This is due to the capacities of the DBSCAN method which allows to detect all the clusters, even if they have a convex shape. Indeed, only the true clusters were taken into consideration by the method, the others are judged as noises.

6.2.1.2 Indexing method

In the indexing fog layer, the used Binary tree with Hyper-plane (BH), similar to the B3CF-tree [280], is based on a recursive division of the space, by an

Algorithm 5 CV method

```
Require: $Cl = \{Cl_1..Cl_n, n = 1..N\}$ $Cl' = \{Cl'_1..Cl'_k, k = 1..K\}$
Ensure: $I_m$
  for each data stream do
    for $cl' \in Cl'$ do
      $CV_{cl'}$ ←Calculate the coefficient of variation of the new cluster $(cl')$
      for $cl \in Cl$ do
        $CV_{cl' \cup cl}$ ← Calculate the coefficient of variation of $(Cl' \cup Cl)$
        if $CV_{cl'} < CV_{cl' \cup cl}$ then
          create new index $(cl')$
        else
          insert $cl'$ in $I_n$
        end if
      end for
    end for
  end for
```

hyper-plane, into two regions through two pivots p_1, p_2 chosen as the two farthest elements. In the set E, elements closer to p_1 belong to the first region while those closer to p_2 belong to the second region. This results in avoiding the overlapping of regions when answering queries. Firstly, a leaf node LN contains a subset E_{LN} of objects with $E_{LN} \subseteq E$. Secondly, an inner node IN consists of two elements and two children: $(p_1, p_2, L, R) \in \mathcal{O}^2 \times \mathcal{IN}^2$. That is :

- p_1,p_2 are two unconfused objects, $d(p_1, p_2) = d_{max}$, called "pivots". They define the hyper-plane.
- L is a left sub-tree and R is a right sub-tree.

The construction of BH-tree is realized incrementally.The insertion is top-down.

6.2.1.2.a Parallel kNN similarity queries search

The parallel kNN method is adopted for the similarity query search in BH-trees because the add of an arrival cluster to the first clusters induce a re-computation of the Delaunay graphs in the cloud and in all fogs which may cause a latency of continuous IoT data indexing process. The search algorithm gives an answer to the query q with radius r_q to recover the k objects closest to q (Algorithm 6). The set of k objects is stored in the list A. To address the queries, we apply the kNN algorithm on the BH-tree by starting from the root to its leaves. The search is performed by calculating the distance between the query point and the two pivots p_1 or p_2, going down the tree and determining whether the search should continue in the left branch L or the right branch R. We start the query with a radius $r_q = +\infty$ and then, decrements by traversing each sub-tree that corresponds to the distance to the k^e object in the order list A. To make the kNN search more efficient, parallelism is also used in this work in the similarity search query process to minimize retrieve time [278]. Indeed, the complexity of the kNN search in all indexes could be reduced to the complexity of search in only one index.

To test the efficiency of our proposed approach, the CV method will be confronted to two other scenarios. For these scenarios, the fog layer contains only the clustering and the indexing levels. The first scenario is called Creation of a New Index (CNI) and the second scenario is called Insertion in an Insertion in an Existing Index (IEI).

6.2.1.2.b CNI method

In this scenario, objects in clusters Cl' of the arrival data stream are indexed in a new BH-tree. The description of this method is presented in algorithm 7. This method is simple and it needs no comparison with the existing clusters or indexes. The CNI method results in the creation of indexes of similar objects.

Algorithm 6 kNN search in the BH-tree

```
kNN-BH-tree $\left( \begin{array}{l} IN \in \mathcal{N}, \\ q \in \mathbb{R}^n, \\ k \in \mathbb{N}^*, \\ d : \mathcal{O} \times \mathcal{O} \to \mathbb{R}^+, \\ r_q \in \mathbb{R}^+ = +\infty, \\ A \in (\mathbb{R}^+ \times \mathcal{O})^{\mathbb{N}} = \emptyset \end{array} \right) \in (\mathbb{R}^+ \times \mathcal{O})^{\mathbb{N}}$
with:
-$(p_1, p_2, L, R) = IN$
- $d_1 = d(p_1, q)$
- $d_2 = d(p_2, q)$
-$B(q, r_q)$ query ball q with radius $r_q$
if $IN == NULL$ then
  return A
else
  Calculate the distances $d_1$ and $d_2$
  if $|A| < k$ then
    $r_q \leftarrow +\infty$
  else
    $r \leftarrow A$
  end if
  for $i \in (0, 1)$ do
    if $d_i < r_q$ then
      $A \leftarrow k - Insert(k, A, (d_i, p_i))$
    end if
    for each node $IN$ do
      if $B(q, rq) \cap IN \neq \varnothing$ then
        $A \leftarrow kNN - BH - tree(IN_i, q, k, d, r_q, A)$
      end if
    end for
  end for
end if
```

Algorithm 7 CNI method

```
Require: $Cl = \{Cl_1..Cl_n, n = 1..N\}$
  $Cl' = \{Cl'_1..Cl'_k, k = 1..K\}$
Ensure: $I_{n+k}$
  for each data stream do
    for $cl' \in Cl'$ do
      create new index( $cl'$)
    end for
  end for
```

6.2.1.2.c IEI method

In this scenario, objects of each cluster of the arrival data stream are inserted in one of the existing indexes. In this method, clusters centers of the first data stream c_n are took as representatives of the existing indexes. The choice of an existing BH-tree, in which, the objects of the arrival cluster Cl' will be inserted is basing on the test of distances between the arrival cluster center c'_k and the existing BH-tree representative centers c_n (Algorithm 8). Objects of the arrival cluster Cl' will be inserted in index n when the distance between c'_k and c_n is minimum.

Algorithm 8 IEI method

```
Require: $Cl = \{Cl_1..Cl_n, n = 1..N\}$
  $Cl' = \{Cl'_1..Cl'_k, k = 1..K\}$
Ensure: $I_n$
  for each data stream do
    for $cl'_k \in Cl'$ do
      for $cl_n \in Cl$ do
        $Min_d \leftarrow$calculate distances($d(c_n, c'_k)$)
        insert $cl'$ in $I_n$
      end for
    end for
  end for
```

6.3 Simulation and Results

In this section, we firstly describe the experimental parameters, including the datasets and the experimental platform. Then, we report and discuss our experimental results with respect to the evolution of the number of indexes with the data stream, the evaluation of the indexes construction and the evaluation of the parallel kNN search.

6.3.1 Experimental setting

For the experimental evaluation of the four proposed indexing methods, we have used three real data sets (GPS trajectory, WARD and traffic datasets) and one synthetic dataset (Tracking). Details on these datasets are presented in what follow.

1. GPS Trajectories: Collected from Go!Track Android application [271].
2. Tracking dataset: Moving vectors generated by an object tracking simulator with wireless cameras in the wireless multimedia sensor network in a random simulation [5].
3. WARD (Wearable Action Recognition Database)[272]: Database of human action reconnaissance using wearable movement sensors [273].
4. Traffic dataset: Belongs to the road networks category [281].

In order to achieve our data stream simulation experiments, all datasets were divided into subsets. These subsets of different sizes and dimensions (Table 6.2) are considered as data streams. Our experiments were implemented using Python software installed in a 64-bit Linux operating system (Ubuntu) of Intel®Core TM i7-8550U CPU, 1.80 GHz*8 processor, 16GB RAM and 256GB ROM.

Table 6.2: Characteristics of the selected datasets for the index evaluation.

Dataset	**Size (Vectors)**	**Dimension**	**Size of the data stream(Vectors)**	**Size of data stream (Bytes)**
GPS trajectory	18107	3	4000	115507.02
Tracking a moving object dataset	62702	20	12000	1270493.8
WARD	3078552	5	600000	18058184
Traffic dataset	5000000	2	1000000	20132659.2

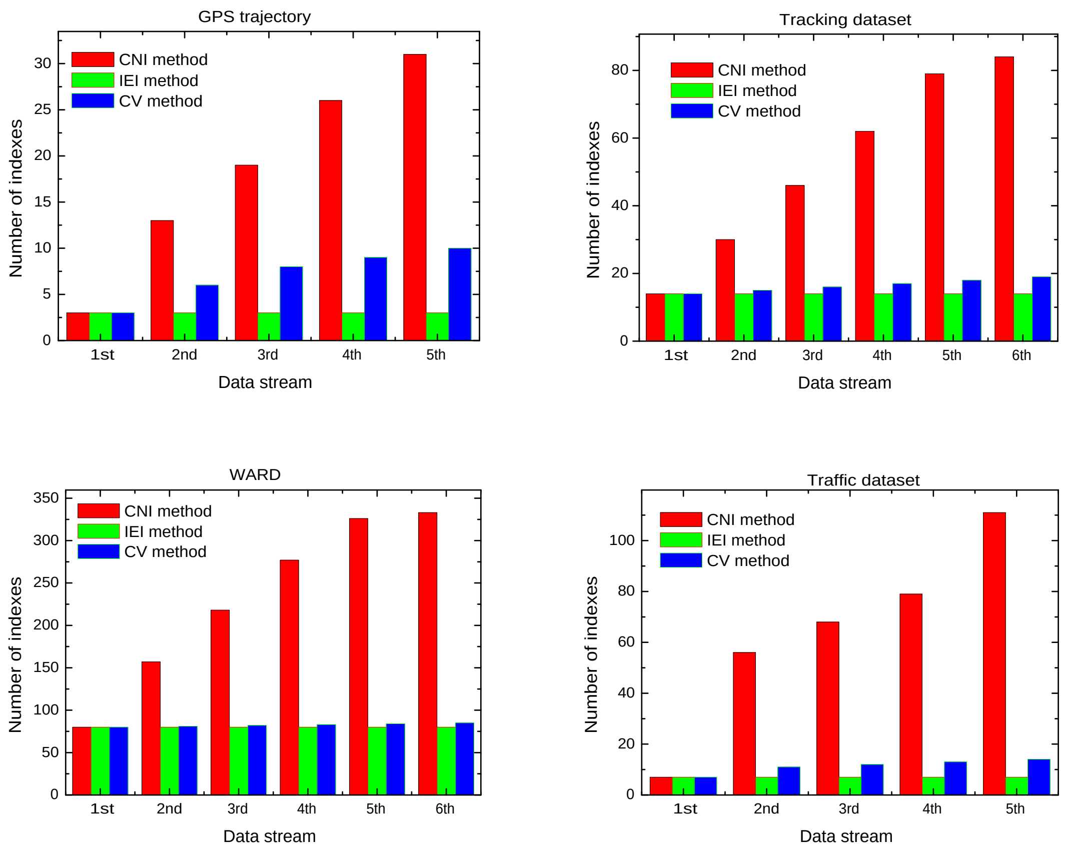


Figure 6.3: Number of BH-trees versus data stream.

6.3.2 Evolution of the number of indexes with the data stream

The variation of the number of indexes as a function of the stream for the used datasets is presented, in figure 6.3, for the CV method and the two other scenarios. It is to notice that for the first data stream, the BH-tree of each cluster was directly constructed. The proposed method is used from the second data stream. As awaited, the use of the IEI method results in the construction of a minimum number of indexes that remains invariant with the data stream. In contrast to the IEI method, the use of the CNI method results in the construction of a maximum number of indexes that increases proportionally to the increasing number of data streams. For the CV method, the number of the constructed indexes is between the number of indexes from the IEI method and that from the CNI method. The number of indexes by the CV method is closer to that by the IEI method, for all datasets, which indicates that in the CV method, the insertion process is more pronounced that the construction process. We can see that the number of indexes, from the CV method, varies from a data to another. This depend, directly, on the the dynamic aspect of the DBSCAN clustering which induced a change of distances between clusters centers for each data stream.

6.3.3 Evaluation of indexes construction

For the evaluation of the BH-tree construction, the number of distances, the number of comparisons, the time of indexing and energy consumption are calculated as a function of data stream.

6.3.3.1 Number of calculated distances

In figure 6.4, the number of distances is traced as a function of the data stream for the three methods. We can see that the number of distances during the construction of the BH-trees starts varying from the second data stream. From this data stream, the number of distances varies, from a method to another, as a function of the data size. For the four datasets, the CNI method presents the highest number of distances, since the creation of pivots requires more distances calculation, while the IEI method presents a less number because in the insertion process, no pivots are created. Despite the CV method combined both insertion and the indexing processes, the number of distances, from this method, is close to that from the IEI method and this reflects the efficiency of the CV method.

6.3.3.2 Number of calculated comparisons

The variation of the number of comparisons, as a function of the data stream, is plotted in figure 6.5. As can be seen, this variation is similar to that of the number of distances in figure 6.4. For the three methods, the number of comparisons is greater than the number of distances. During the construction of new indexes or during the insertion, comparisons are required to choose the left side or the right side of each BH-tree.

6.3.3.3 Time of indexing

As shown in figure 6.6, the time of indexing depends, not only on the data stream, but also on the size of each data stream. For GPS trajectory data, the time of indexing is great when the IEI method is used while for tracking and WARD data, the time of indexing is maximum when the CNI method is used. For the traffic dataset, the time of indexing varies from a method to another, as a function of the data stream. For the four used datasets, the indexing of data streams using the

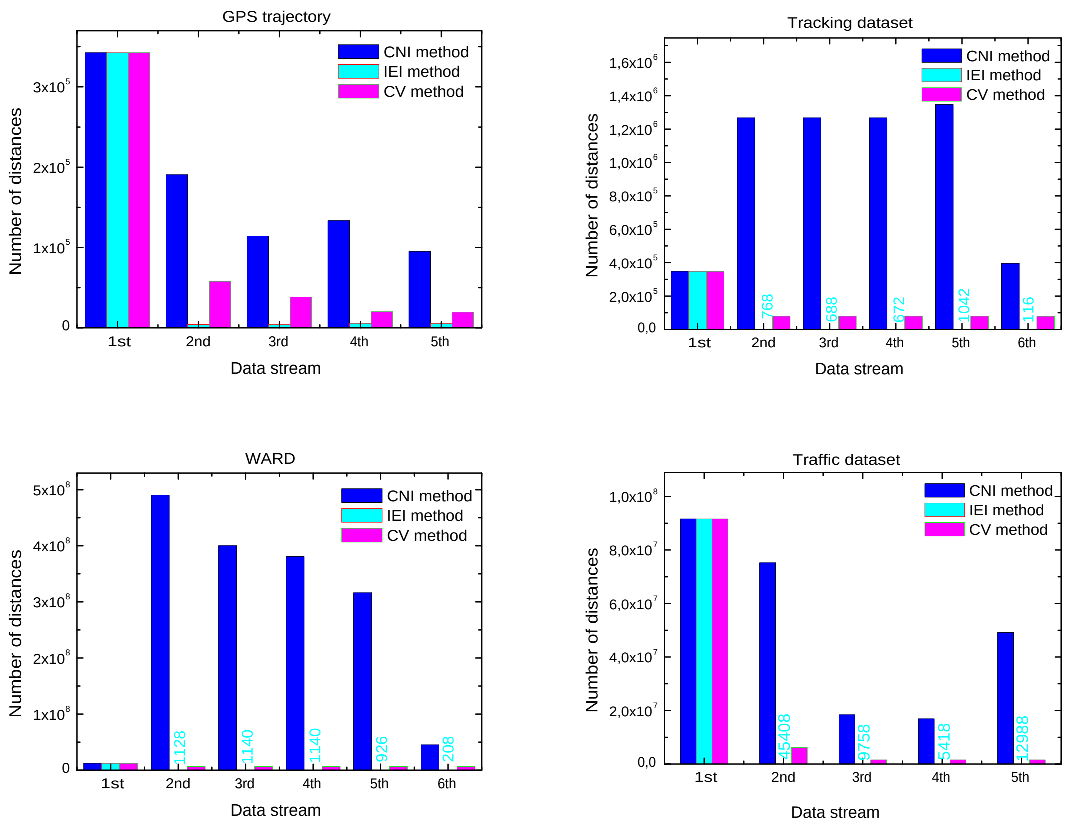


Figure 6.4: Number of distances calculated during the indexing of each data stream.

CV method takes acceptable times whatever the size of the data stream. Contrary to the IEI and the CNI methods, the CV method is not sensitive to the size of the data stream.

6.3.3.4 Energy consumption during the indexing

The energy consumption per stream is traced, in figure 6.7, for CV, CNI and IEI methods. The energy consumption E_{prog} (in Joule) during the execution of a program *prog* is given by the following expression [282] :

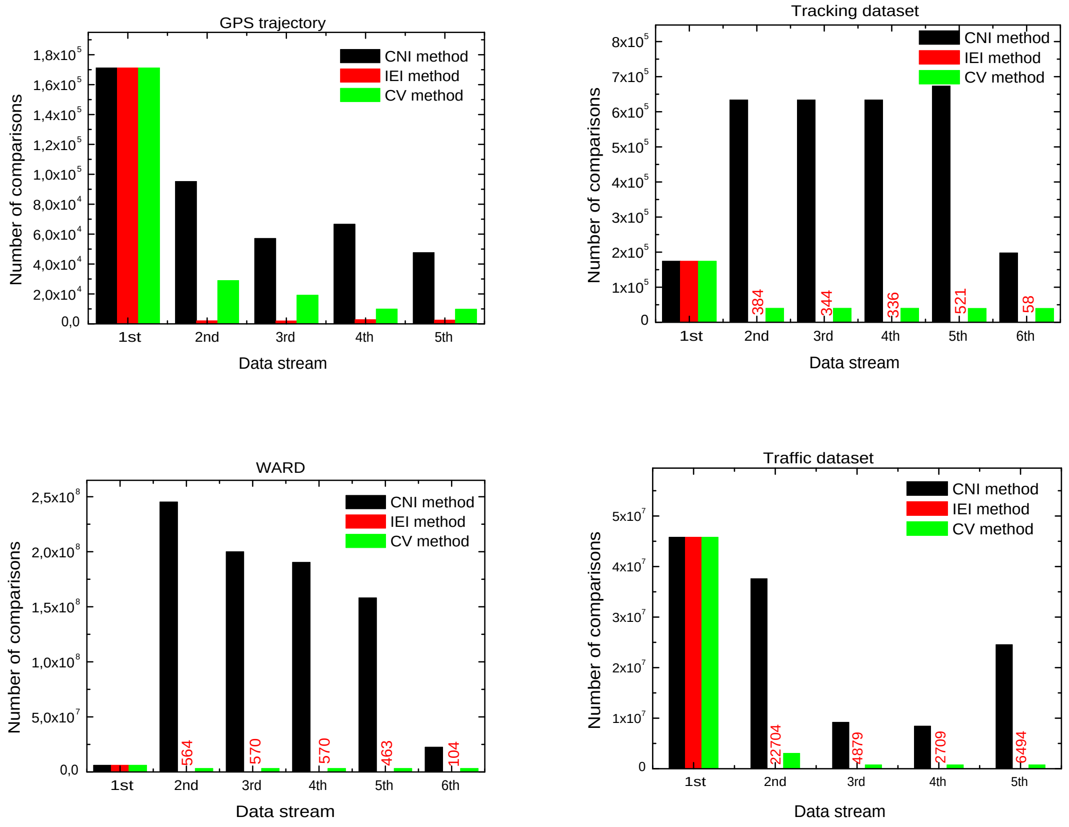


Figure 6.5: Number of comparisons calculated during the indexing of each data stream.

$$E_{prog} = \int_{t_b}^{t_e} P(prog, t)dt - \int_{t_b}^{t_e} P_i(t)dt \tag{6.1}$$

where t_b and t_e the beginning time and the end time of the execution of the program *prog* (in second), $P(prog, t)$ the electrical power needed for the execution of the program *prog* (in Watts) and $P_i(t)$ the idle power (in Watts). As can be seen in figure 6.7, the energy consumption during the indexing using these three methods varies from a dataset to another. For the GPS trajectory, the energy consumption is elevated during the data indexing using the IEI method. The energy consumption during the use of the CV method is a little bit less than

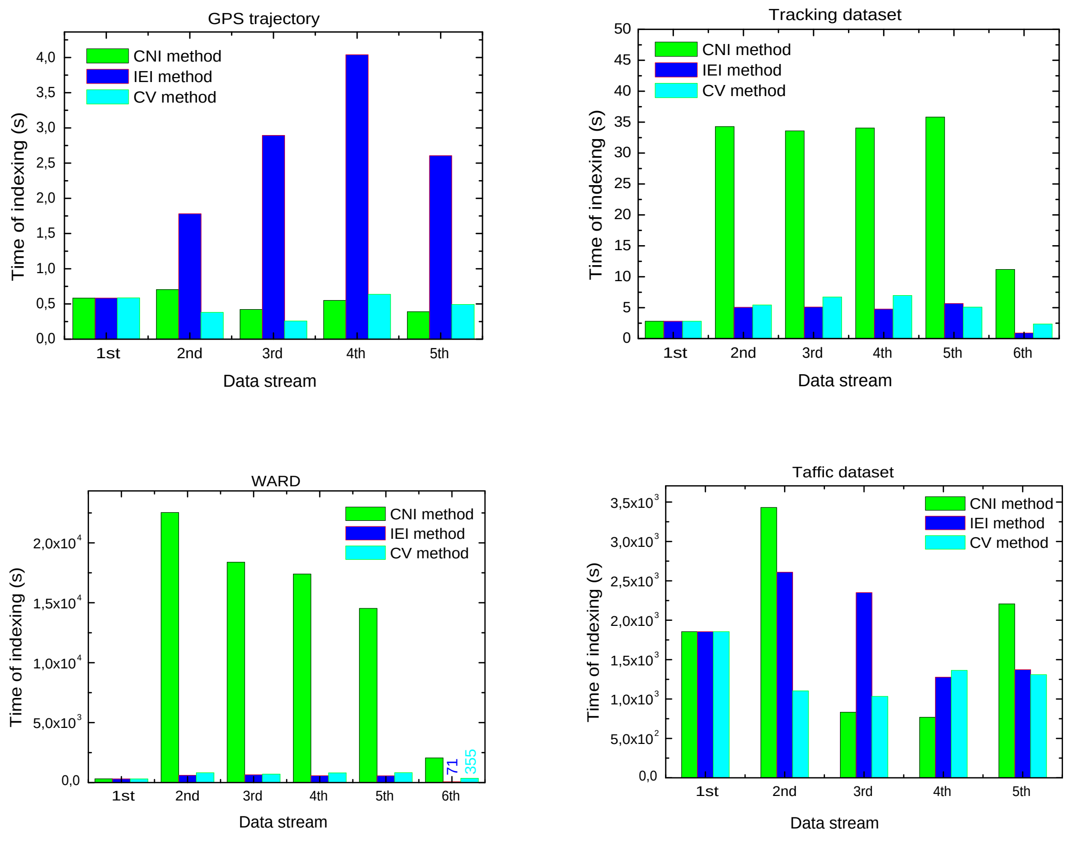


Figure 6.6: Time of data stream indexing using the CV method compared with both the IEI and the CNI methods.

that during the use of the CNI method. Contrary to the GPS trajectory dataset, the energy consumption during the indexing of both tracking and WARD datasets using the CNI method is higher compared with the CV and the IEI methods. These last are mainly close. For the traffic dataset, the energy consumption during the indexing using the CNI and the IEI methods is comparable and is greater than that during the use of the CV method.

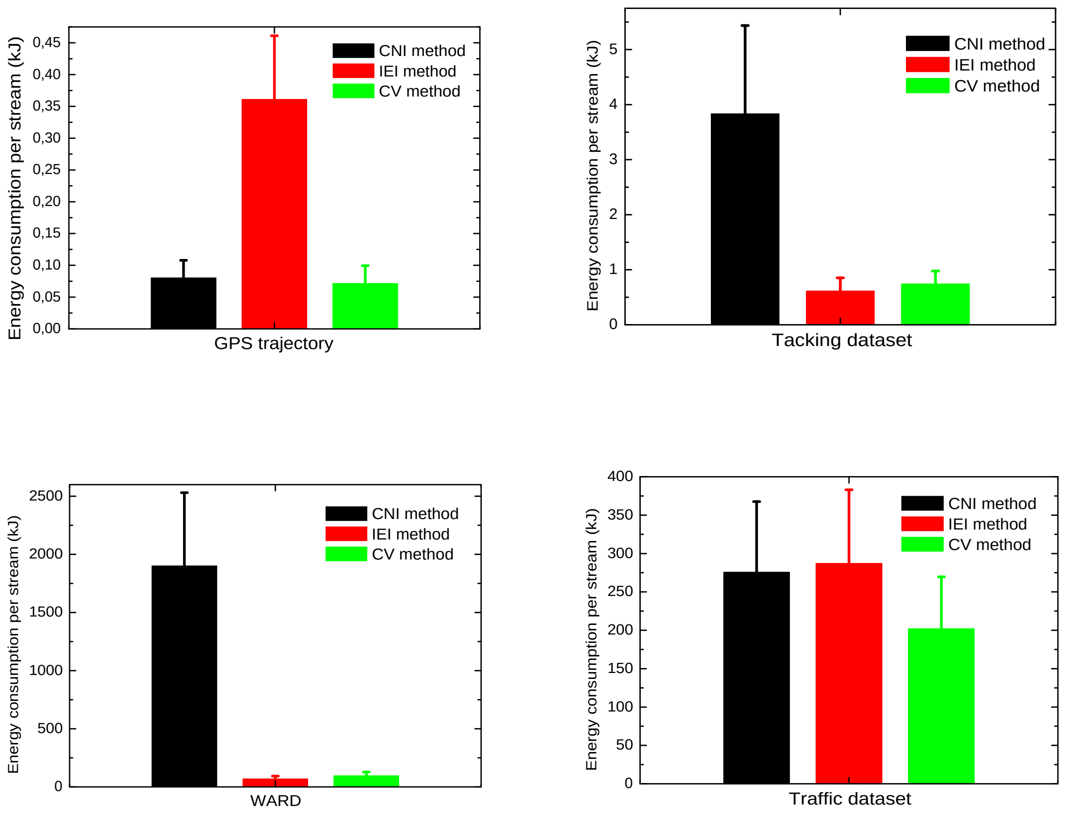


Figure 6.7: Average energy consumption during indexes construction using CV, CNI and IEI methods.

6.3.4 Quality of the constructed BH-trees

For the evaluation and the comparison of the quality of BH-trees constructed using the CV method with those from the IEI and the CNI methods, the average height of indexes, the average number of internal nodes and the average number of leaves nodes are plotted, for the four datasets, in figure 6.8. The number of nodes per level (Figure 6.9) and the data distribution in leaves (Figure 6.10) are determined after the indexing of all data streams.

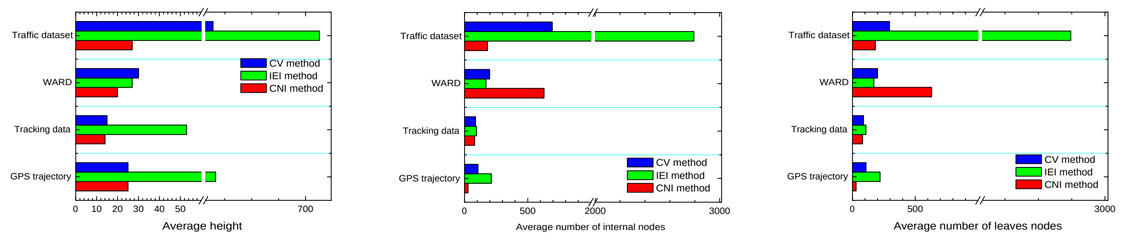


Figure 6.8: Average height, average number of internal nodes and average number of leaves nodes of BH-trees constructed using the CV method, the CNI method and the IEI method.

6.3.4.1 Average height of BH-trees

Figure 6.8 presents the average height of BH-trees resulting from the indexing of streams of GPS trajectory, tracking, WARD and traffic datasets. For all datasets, the average height of indexes constructed using the IEI method is greater than that of indexes constructed using CNI method. This is due to the fact that in the IEI method, all data are inserted in constant number of BH-trees while in the CNI method, for each cluster, a BH-tree is constructed. We can also see, in figure 6.8, that the average height of indexes constructed using the CV method is comparable to that of the CNI method for the GPS trajectory and the tracking datasets while for the WARD and the traffic datasets, the average height from the CV method is greater than that for the CNI method and slightly surpasses the average height from the IEI method for the WARD data. The CV method changes its behaviors as a function of the data stream size and dimension. It behaves like the CNI method when indexing the GPS trajectory and the tracking datasets and like the IEI method when indexing the WARD and the traffic datasets.

6.3.4.2 Average number of internal nodes

The average number of internal nodes per BH-tree varies from a dataset to another as can be seen in figure 6.8. For GPS trajectory, tracking and traffic datasets, the average number of internal nodes in indexes constructed using IEI method is greater than that in indexes constructed using CNI method contrary to the WARD dataset where the average number of internal nodes in BH-trees constructed using the CNI method is greater than that in indexes by the IEI method. For all datasets the average number of internal nodes constructed using the CV method is located between those of the CNI and IEI method. As awaited, the variation of the average number of leaves nodes, as a function of the indexing method, is similar to that of the average number of internal nodes (Figure 6.8).

6.3.4.3 Number of nodes per level

The number of nodes per level in BH-trees constructed using CNI, IEI and CV methods is traced in figure 6.9 for the four used datasets. As can be seen, the number of nodes per level varies from the a dataset to another. For the GPS trajectory, the number of nodes is constant in all levels of the BH-tree whatever the proposed indexing method. For tracking dataset, the variation of the number of internal nodes per level changes as a function of the indexing method. For the IEI method, five levels contain an elevated number of nodes while two levels with a maximum number of nodes are obtained from the CV method and only one level of maximum nodes is obtained from indexes by the CNI method. For the WARD dataset only one level with maximum number of nodes in indexes constructed using both CNI and IEI methods. The indexes constructed using the CV method contain two levels of maximum number of nodes. For the traffic dataset, one level with maximum number of node is obtained from BH-trees constructed using the CNI and the CV methods. For the IEI methods, three levels have high number of nodes. The number of nodes remains constant beyond level 25 for the CV mathod (2 nodes per level) and beyond level 60 for the IEI method (25 nodes per level).

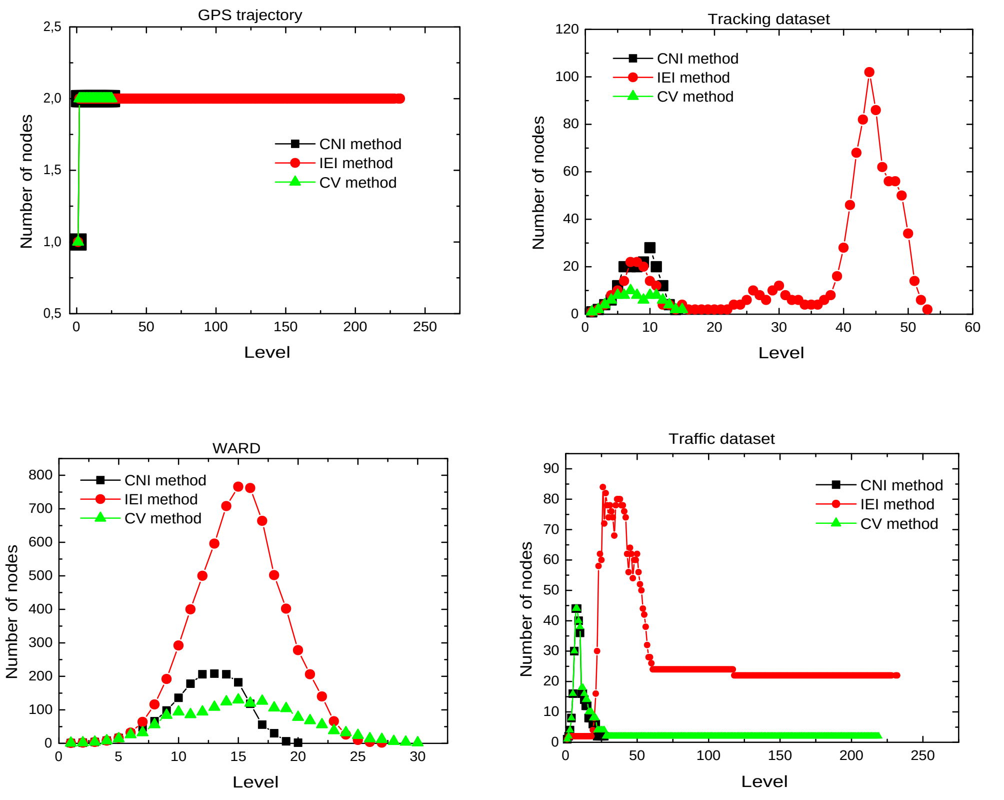


Figure 6.9: Variation of the number of nodes per level of BH-trees constructed using the CNI, IEI and CV methods.

6.3.4.4 Data distribution in BH-tree leaves

The distribution of data in the left and the right sides of the BH-tree is plotted, in figure 6.10, for CNI, IEI and CV methods. For GPS trajectory dataset, resulting indexes constructed using both CNI and IEI methods are not balanced while indexes constructed using CV method are well balanced. For the trajectory, WARD and traffic datasets, indexes from the three proposed methods are well balanced. We can also see that the data distribution in indexes constructed using the CNI and IEI methods is similar whatever the used dataset.

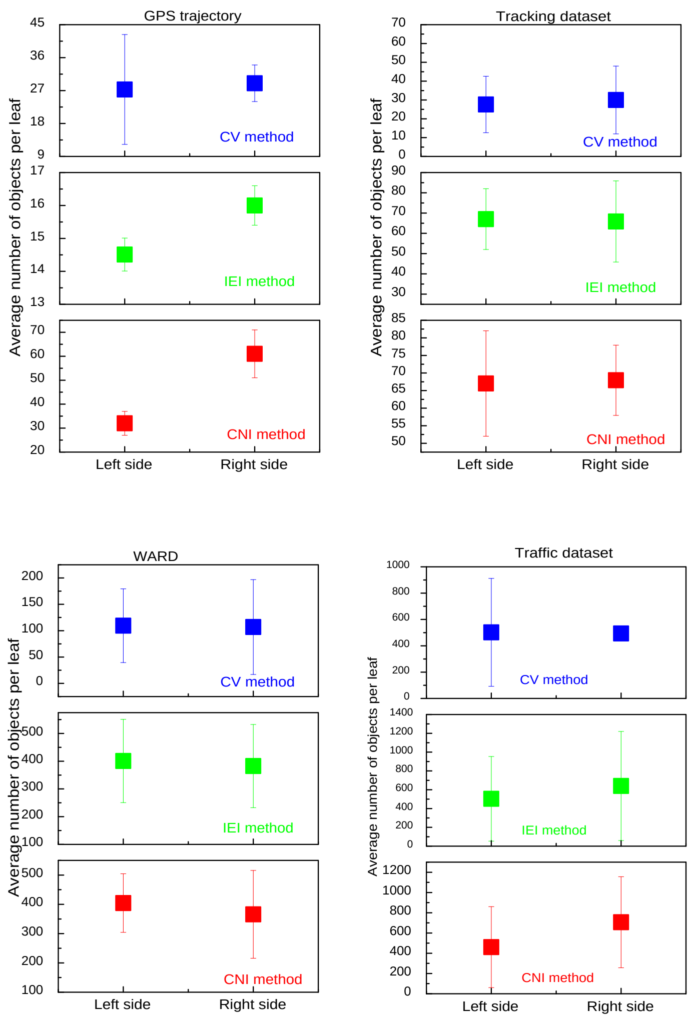


Figure 6.10: Data distribution in leaves.

6.3.5 Evaluation of the parallel kNN search in BH-trees

For the evaluation of the parallel kNN search with $k = 5, 10, 15, 20, 50$ and 100 in BH-trees constructed using CNI, IEI and CV scenarios, the number of distances, the number of comparisons, the time of search, energy consumption and the number of visited leaves is determined to search 100 queries. It is to notice that all statistical results were averaged over 100 randomly generated queries.

6.3.5.1 Number of calculated distances

The average number of calculated distances during the kNN search with $k = 5, 10, 15, 20, 50$ and 100 is plotted, in figure 6.11, for the three methods. As can be seen in figure 6.11, the average number of distances varies from a data to another. For the GPS trajectory dataset the average number of distances during the querying search in BH-trees constructed using CNI and CV methods are close and less than that calculated during the query search in indexes constructed using the IEI method. This could be correlated with the variation of the average height of indexes (Figure 6.8) since the number of nodes per level is unvaried for the GPS trajectory data (Figure 6.9). For tracking and WARD data sets the average number of distances calculated during the kNN query search in indexes constructed using the CV method is less than the number of distances in indexes constructed using CNI and IEI methods.

This can be related to the variation of number of nodes per level (Figure 6.9). For the tracking dataset and for levels between 5 and 10 the number of nodes from the CNI method is greater than that from IEI method and that of IEI method is greater than that the CV method. For the WARD data set, the number of nodes per level from the IEI method is greater than that from the CNI method which is greater than the number of nodes per level from CV method. For the traffic dataset, the number of distances calculated during the query search in indexes constructed using the CV method is greater than that in indexes by the CNI method and less than that in indexes by the IEI method. This could be directly related to the indexes height (Figure 6.8).

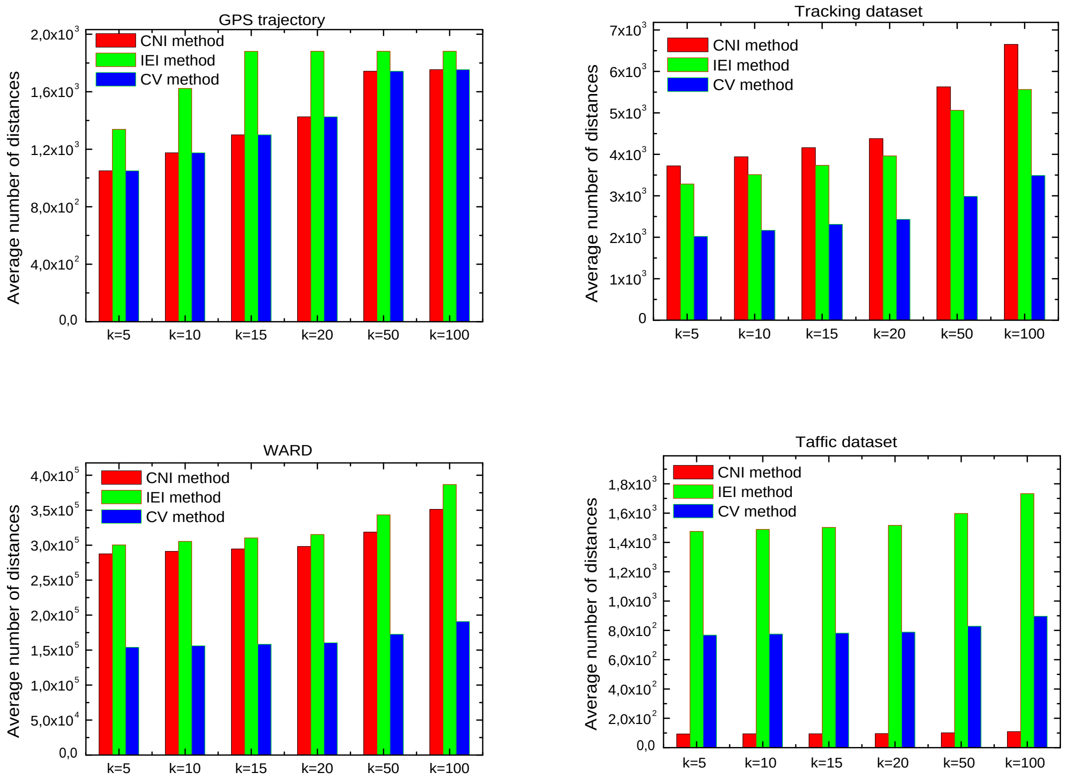


Figure 6.11: Number of distances calculated for the kNN search in BH-trees by CNI, IEI and CV methods.

6.3.5.2 Number of calculated comparisons

The average number of comparisons calculated during the kNN queries search in BH-trees constructed using CNI, IEI, CV methods is presented in figure 6.12. A similar variation is observed for the four dataests.

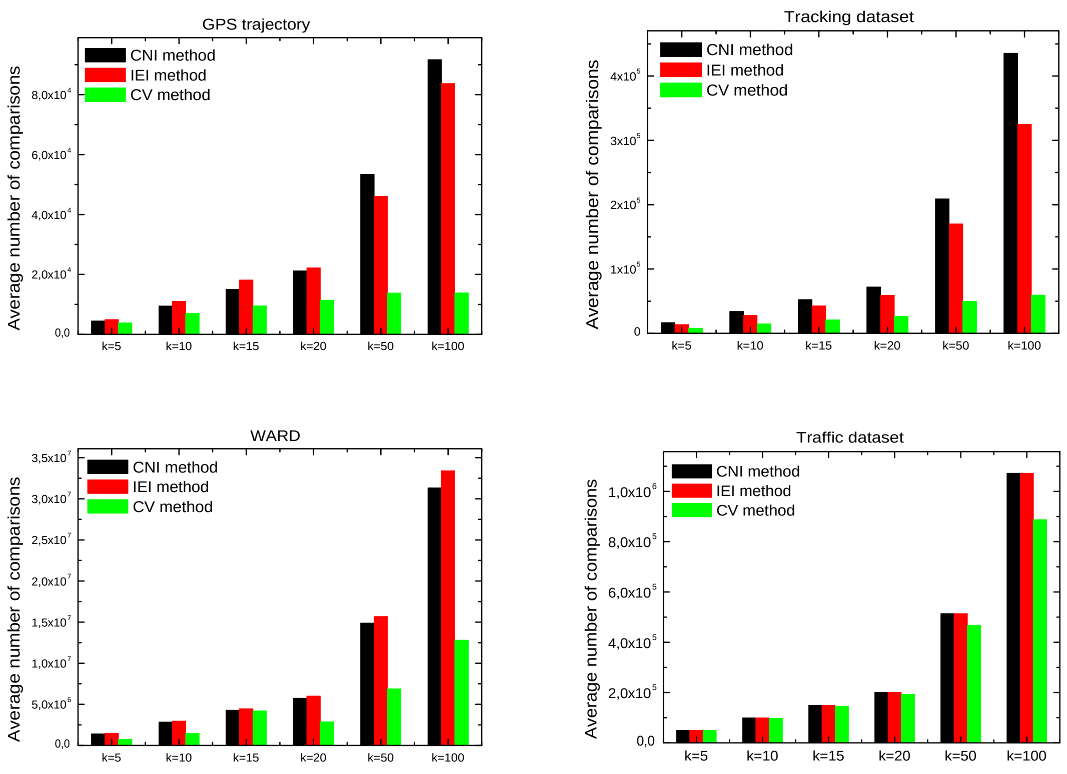


Figure 6.12: Number of comparisons calculated for the kNN search in BH-trees by CNI, IEI and CV methods.

We can see that the average number of comparisons, calculated in indexes constructed using the CV method, is less than that in indexes constructed using CNI and IEI methods. This may due to the fact that the use of the CV method results in fusion of clusters of similar objects in contrast to the IEI method in which, heterogeneous objects are inserted in constant number of indexes.

6.3.5.3 Time of search

Figure 6.13 shows the variation of the time of kNN search queries in BH-trees constructed using CNI, IEI and CV methods. The variation of the time of search for the three scenarios is related to the variation of both the average number of distances (Figure6.11) and the average number of the visited leaves (Figure 6.14).

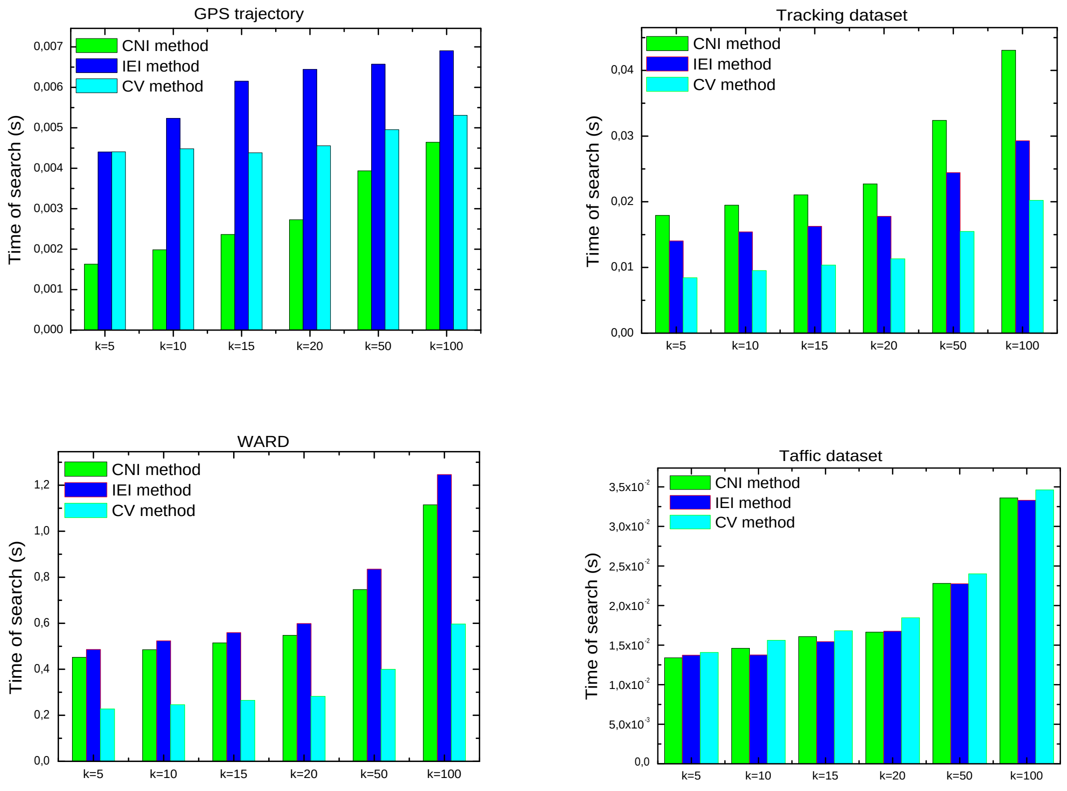


Figure 6.13: kNN search time of CNI, IEI and CV method.

For the GPS trajectory dataset, the shortest time of search is obtained for indexes constructed using CNI method where the time of search varies from 0.0016 to 0.0046s when k varies from 5 to 100. For this indexing method, the number of both the distances and the visited leaves are less then those of the two other methods. The time of search in indexes constructed using the CV method is nearly invariant as a function of k and always located between that of both CNI and IEI methods. The time of search in indexes by the CV method is around 0.0048s. For tracking and WARD datasets, the CV method presents the shortest time of kNN query search which varies from 0.008 to 0.02s for the tracking dataset and from 0.227 to 0.596s for the WARD dataset when k varies from 5 to 100. For the traffic dataset, despite the minimum of the time of search correspond to the IEI method indexes, the search time for the three methods is close. It varies around 0.0137 and 0.0338s when k varies from 5 to 100. The time of search for the traffic dataset is less than that for the WARD data because the number of visited leaves for the traffic dataset is less than that for the WARD dataset as can be seen in figure 6.14. For $k = 100$ and for tracking data, the time of search in indexes by the CV method is 46% of that by the CNI method and 69% of that by the IEI method while for the WARD data, it is 53% and 47% of that by the CNI and the IEI respectively.

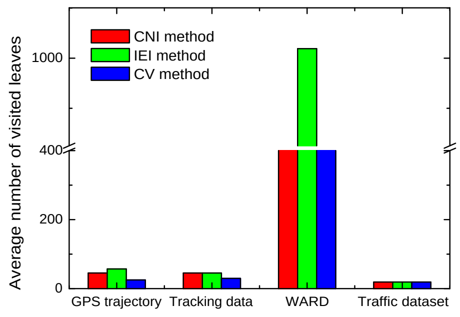


Figure 6.14: Number of the visited leaves in CNI, IEI and CV method.

For traffic dataset, the time of search in indexes by the IEI method is 96% of that by the CV method and the time of search for the CNI method is 97% of that by the CV method. In the CV method, when the coefficient of variation (CV) of the union of a new cluster from the incoming data stream with the existing first clusters is minimum this means that objects in the two clusters are very similar. Thus, objects, in the new cluster, are inserted in the index corresponding to the first cluster which make objects in this index more similar. That is why, during the kNN query search in the CV method indexes, the number of distances and the number of visited leaves are less compared with the other methods. For the GPS trajectory dataset, the height of indexes influenced directly the search time because indexes constructed using CV, CNI and IEI methods have a constant number of nodes per level (Figure 6.9). In ref.[5], Benrazek et al. indexed the whole above cited datasets in a BCCF-tree. For $k = 100$,they found 0,16191, 0,21034 and 2,72482s for GPS trajectory, tracking and WARD datasets respectively. For $k = 100$, the time of search, obtained by Zhang et al. [283], which is 0.682s for a data of 1 million is comparable to that of the WARD dataset of 3 millions size. The improvement of the time of search using our proposed methods came from the use of DBSCAN clustering algorithm witch results in the creation of clusters of high similarity.

6.3.5.4 Energy consumption during the kNN search

Figure 6.15 presents the energy consumption during the parallel kNN search with $k = 100$ in indexes constructed using CV, CNI and IEI methods. For the four selected datasets, the CNI method consumes energy more than CV and IEI methods. That was awaited since the use of the CNI method induces the creation of more indexes. In addition, the use of parallelism induces an energy consumption in all indexes. The energy consumption during the 100NN search in indexes constructed using the CV method is comparable to that for the IEI method which reflects its efficiency during the indexing of continuous data streams.

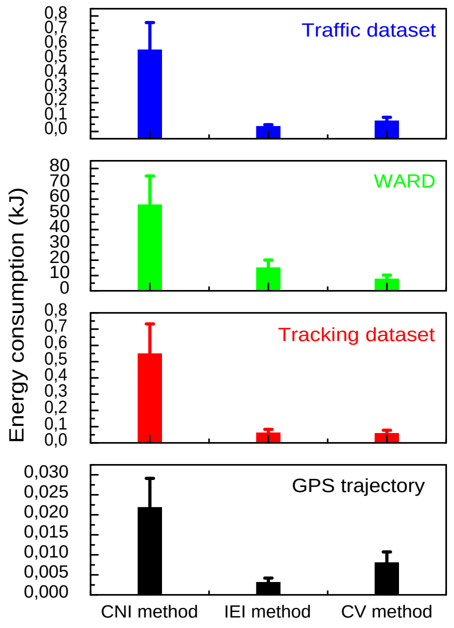


Figure 6.15: Energy consumption during the 100NN search for CNI, IEI and CV method.

6.4 Conclusion

According to the comparison of results we conclude, although the kNN search time in indexes constructed using the CNI and the IEI methods is comparable with that of the above-cited methods, some of their characteristics are not desirable for continuous IoT data stream indexing. The CNI method is capable of dynamically index the continuous IoT data stream; it produces a very large number of indexes. The construction of these indexes is a high cost process in terms of the number of distances, the number of comparisons, the time computing and the energy consumption. The large number of indexes increases also, the cost of the number of distances, the number of comparisons, the time and the energy consumption of search computing during the kNN query search. In addition, creating a large number of indexes leads to the risk of memory overload, which negatively affects the indexing process of the continuous IoT data. The IEI method allows indexing the continuous IoT data stream with low computational cost of the number of distances, the number of comparisons and the time. However, it is not adapted to continuous IoT data stream. The inclusion of large various elements, from the continuous incoming data, in a limited number of indexes increases the height of these last and reduces their similarity since the criterion of insertion is the minimum distance between the existing and the incoming clusters centers. Increasing the depth of indexes and reducing their similarity may lead to the increase of the number of distances, the number of comparisons and the time of kNN search computations. In addition, inserting large number of elements faces indexes, constructed using the IEI method to the problem of degradation. Compared with the CNI and the IEI methods, the CV method presents more capability to, dynamically, index the continuous IoT data stream. The coefficient of variation (CV) determines whether the resulting cluster from the union of the existing and the incoming clusters is similar or dissimilar. If the union of clusters is similar, incoming elements are inserted in the corresponding existing cluster index and, if the union of clusters in dissimilar, a new index is constructed from the incoming cluster. Having similar elements in each index reduces the cost of the computing of the number of distances, the number of comparisons, the energy consumption and the time of kNN query search. In addition, creating new indexes, in the case where the union of clusters is not similar, makes this method an appropriate approach for indexing continuous IoT data stream. By using the CV method, we avoid the problem of the infinite number of indexes and the degradation of the indexes and we guarantee the construction of new indexes with low energy consumption and without data overlap.

7 TD Method for Indexing Continuous IoT Data

7.1 Introduction

In the previous chapter, we presented a new approach to index the continuous IoT data stream using the Coefficient of Variation (CV) method. We propose, in this chapter, a new method to index continuous IoT data flow taking the benefit from the fog-cloud architecture. The proposed method, called Threshold Distance (TD) is a two step process. The first process consists on the grouping of the arrival data flow into homogeneous clusters by means of the DBSCAN algorithm [119]. The second process is the construction of GHT (Generalized Hyperplane Tree) [220],[262] for each cluster. The clusters of the first data flow are directly indexed while for those of the next data flows, they will be indexed or inserted in existing GHT after comparing the distances between their centers and the existing clusters centers to a threshold distance value. To test the efficiency of this proposed method, the experimental results will be compared to those of our second proposed method, in this chapter, called Creation of a New Tree (CNT) in which, a new GHT is constructed for each arrival cluster.

The present chapter starts with a detailed description of the proposed approach followed by the exposition of the experimental results of the GHTs construction and those of the kNN query search. The experimental results of both the GHTs construction and the kNN query search will be discussed and compared with those of the CNT index.The experimental results of the kNN search will be also compared with those of the CV method presented in the previous chapter.In the last, The experimental results of the kNN search will be also compared with those of the B3CF-trre method presented in the previous chapter.

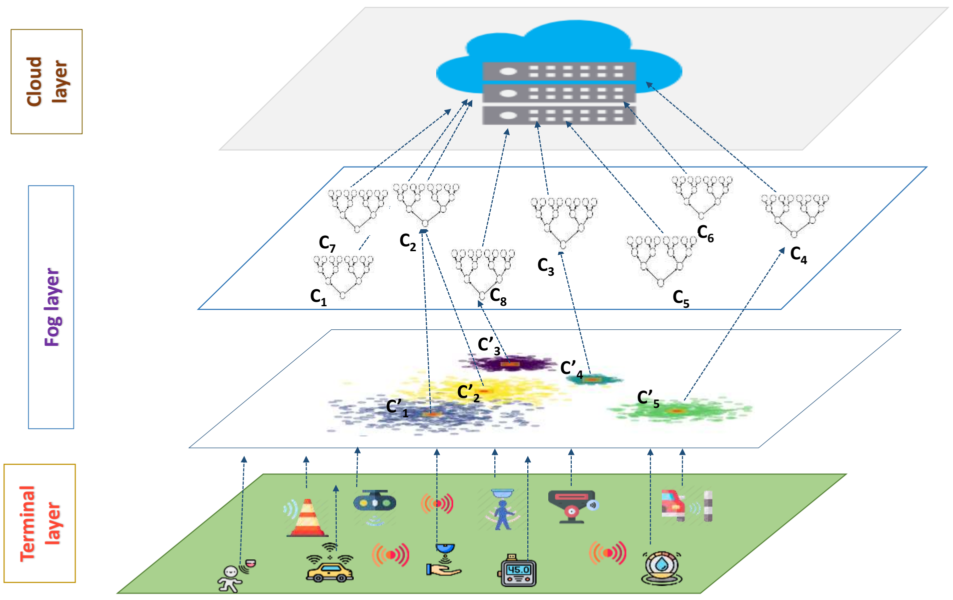


Figure 7.1: Architecture of TD method.

7.2 Proposed Approach

In an IoT environment, data is continuously sent from multiple devices to data warehouses in the cloud. In this approach, before sending data directly to the cloud, we first process it in the fog layer. Because our proposed approach is a two step process, the fog layer is divided into two levels: the clustering fog level and the indexing fog level (Figure 7.1).

The first process, which takes place in the clustering fog level, consists on the grouping of each data flow Fl, of center C_{Fl} into n clusters using the DBSCAN algorithm [119] This results in clusters of high homogeneity and high density with centers C_n (Algorithm 9). In the second step, in the indexing fog level, a GHT (Generalized Hyperplane Tree) [220],[262] is constructed for each cluster Cl of the first data flow. For the next data flows $Fl^{'}$ of a center $C^{'}_{Fl^{'}}$, each resulting cluster $Cl^{'}$ of a center $C^{'}_{n^{'}}$ will be indexed or inserted in an existing GHT after the comparison of the minimum distances between $C^{'}_{n^{'}}$ and the centers of the first clusters $d_{min}(C_n, C^{'}_{n^{'}})$ to the threshold distance value TD. The threshold distance TD is determined as the average distance between the next data flow center $C^{'}_{Fl^{'}}$ and the centers of the first data flow clusters C_n. If $d_{min}(C_n, C^{'}_{n^{'}}) > TD$, a new GHT is constructed and the cluster center $C^{'}_{n^{'}}$ is added to the first data flow clusters centers C_n. If $d_{min}(C_n, C^{'}_{n^{'}}) \leq TD$, object of the cluster $Cl^{'}$ are inserted in the GHT which correspond to the cluster center C_n.

Algorithm 9 TD method

```
Require: $Fl$ //Data flow
Ensure: $GHTs$
  // Clustering of the first data flow
  if $Cl = \emptyset$ then
    $(n, Cl_n) \leftarrow DBSCAN(Fl)$
    //$n$:number of clusters. $Cl_n$: set of clusters $cl$
    for $cl \in Cl_n$ do
      $C_n \leftarrow$Calculate center($cl$)
      //$C_n$: set of clusters centers $c_n$
    end for
    // Indexing of the first clusters $Cl_n$
    for $cl \in Cl_n$ do
      $GHT \leftarrow Build(cl)$
    end for
  else if $Cl \neq \emptyset$ then
    $Fl' \leftarrow Fl$
    //$Fl'$: next data flow
    //Calculation of the center of the next data flow
    $C'_{Fl'} \leftarrow$Calculate center(Fl$'$)
    //Clustering of the next data flow
    $(n', Cl'_{n'}) \leftarrow DBSCAN(Fl')$
    //$n'$:number of next clusters. $Cl'_{n'}$: set of next clusters $cl'$
    for $cl' \in Cl'_{n'}$ do
      $C'_{n'} \leftarrow$Calculate center($cl'$)
// $C'_{n'}$: set of next clusters centers $c'_{n'}$
    end for
    //Calculation of the threshold distance $TD$
    TD$\leftarrow$mean(distance($C'_{Fl'}$,$C_n$))
    for each $cl' \in Cl'_{n'}$ do
      for each $cl \in Cl_n$ do
        if $d_{min}(c_n, c'_{n'}) > TD$ then

          $GHT \leftarrow Build(cl)$
          Add $cl'$ to $Cl_n$
        else
          Insert objects of $cl'$ in $GHT(c_n)$
        end if
      end for
    end for
  end if
```

To check the efficiency of the proposed TD method, the experimental results, in the next section, will be compared to another proposed method extracted from the TD method. We call this method Creation of a New Tree (CNT) and it is also developed in the same fog-cloud architecture. In this method, as its name indicates, a new GHT is constructed for every arrival cluster. The CNT method is described in the algorithm 10.

Algorithm 10 CNT method

```
Require: $Fl$ //Data flow
Ensure: $GHTs$
  // Clustering of the first data flow
  $(n, Cl_n) \leftarrow DBSCAN(Fl)$
  //$n$:number of clusters. $Cl_n$: set of clusters $cl$
  for $cl \in Cl_n$ do
    $GHT \leftarrow Build(cl)$
  end for
  $Fl' \leftarrow Fl$
  //$Fl'$: next data flow
  $(n', Cl'_{n'}) \leftarrow DBSCAN(Fl')$
  //$n'$:number of next clusters. $Cl'_{n'}$: set of next clusters $cl'$
  for $cl' \in Cl'$ do
    $GHT \leftarrow Build(cl')$
  end for
```

7.3 Simulation and Results

Experiments were implemented using Python programming language installed on an Intel® CoreTM i7-8550u, 1.80 GHz processor*8 with 16 GB RAM, 256 GB SDD ROM under 64-bit Linux operating system (Ubuntu). To evaluate our proposed TD method by comparing with our CNT method, two real datasets were used. The tracking dataset contains 6.27k vectors of dimension 20 [5] and the smart home data contains 10M vectors of dimension 4 [275]. For the simulation of data flows, both datasets were divided into subsets considered as flows. The characteristics of these data flows are regrouped in Table 7.1.

Table 7.1: Characteristics of the used data flows.

Dataset	**Dimension**	**Number of flows**	**Vectors per flow**
Tracking dataset	20	6	1.04k
Smart home	4	5	2M

For the experimental results, the evolution of the number of GHTs, the evaluation of GHTs construction and the evaluation of the parallel kNN search in these indexes will be presented and analyzed, as function of the data flow, for the proposed TD method. The results from our CNT method will be used for the comparison with the TD method.

7.3.1 Evolution of the number of GHT

Figure 7.2 presents the evolution of the number of the constructed GHT as a function of the data flow. We can see that the number of GHT is mainly constant when using the TD method and increases proportionally to the data flow when using the CNT method. The use of the TD method results in the construction of 17 GHT for tracking dataset and 7 GHT for smart home data while for the CNT method, 84 GHT were constructed for the tracking dataset and 35 GHT for the smart home data. This indicates that the TD method reduces considerably the number of constructed indexes when compared with the CNT method.In CNT method,with each new data flow, new GHT are produced, meaning that their number is constantly increasing, but In TD, despite the increase in the number of data flow, the number of indexes remains stable.

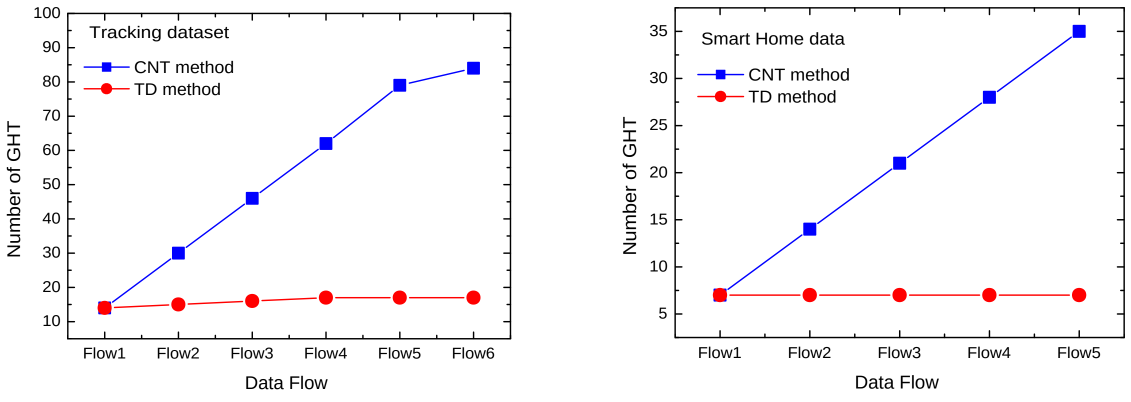


Figure 7.2: Evolution of the number of GHT as a function of the data flow.

7.3.2 Evaluation of GHT construction

For the evaluation of GHT construction, the computed distances, the computed comparisons and the construction cost will be presented as a function of the data flow.

7.3.2.1 Computed distances

From the second flow, and for both used datasets, the computed distances of the CNT method are much higher than those of the proposed TD method (Figure 7.3). This is due to the fact that, during index construction, many distances are calculated for pivots determination and also, for objects inserted in the left or in the right side. This is not the case for objects insertion since there is no pivots calculation.

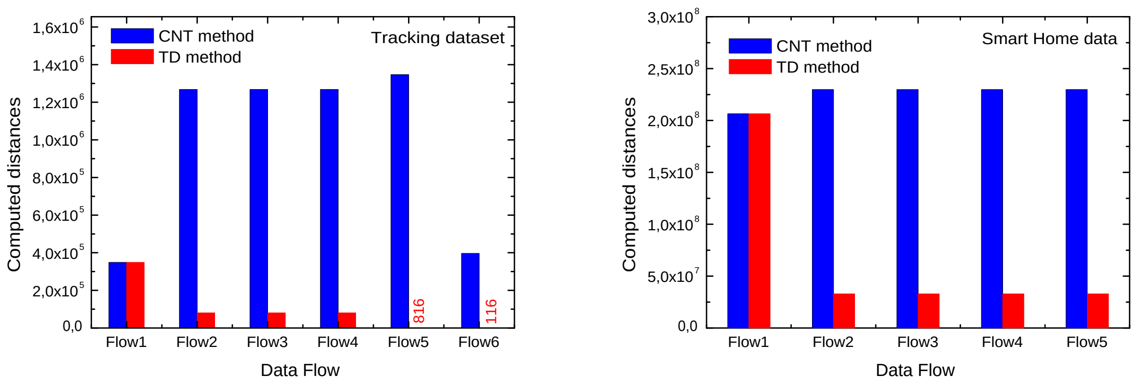


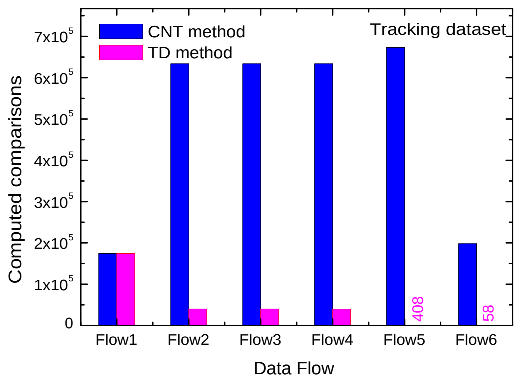
Figure 7.3: Computed distances the TD method and the CNT method as a function of the data flow.

7.3.2.2 Computed comparisons

As for the computed distances, the computed comparisons for both datasets are greater in the CNT method than those in the TD method (Figure 7.4). This reflects the efficiency of the TD method since it combines the insertion process and the construction process depending on the threshold distance value. For the fifth and the sixth tracking data flows, the lowest computed comparisons using the TD method indicates that the whole objects were inserted in existing GHT.

7.3.2.3 Computing time

Figure 7.5 presents the computing time of the TD and CNT methods, for both datasets, as a function of the data flow. For the tracking dataset, the computing time of the TD method is better than that of the CNT method while for the smart home data, the situation is unversed, the computing time of the CNT method is better than that of the TD method in spite of the good results of the TD method concerning the computed distances (Figure 7.3) and the computed comparisons (Figure 7.4). Two parameters influence considerably the obtained results: the



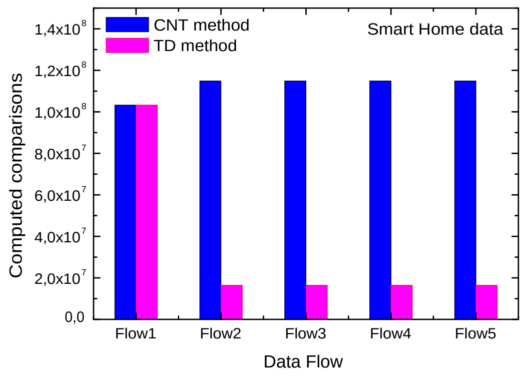
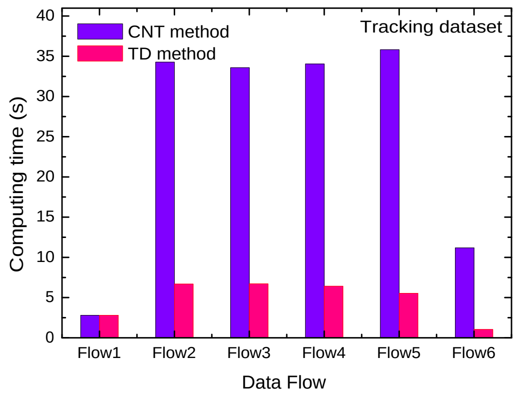


Figure 7.4: Computed comparisons for the TD method and the CNT method as a function of the data flow.



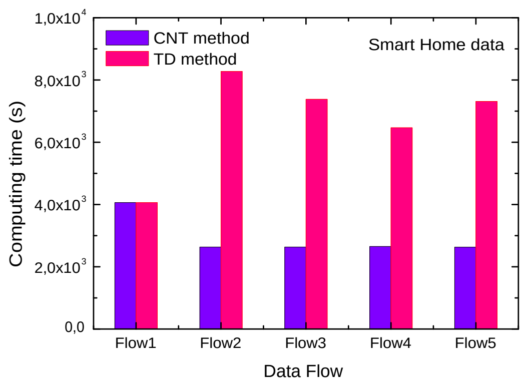


Figure 7.5: Computing time for CNT method and TD method for each data flow.

number and the mean height of indexes. In figure 7.6 is traced the mean height of indexed, for both datasets, constructed using the TD method and the CNT method. As can be seen, in figure 7.6-a, the mean height of GHT constructed using the TD method is greater than that of GHT constructed using the CNT method for both datasets. However, what makes the difference is the global number of GHT as it is shown in figure 7.6-b. Indeed, according to the figure 7.6-b, the

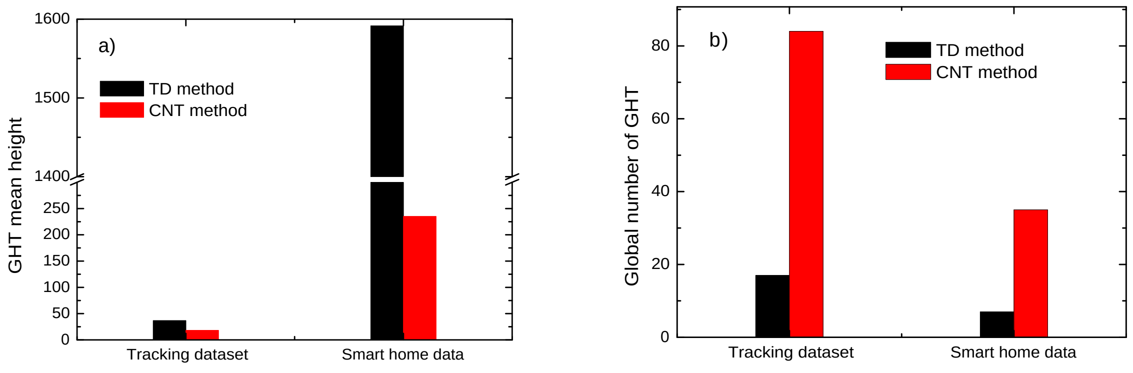


Figure 7.6: Mean height (a) and global number of GHT (b), for both used datasets, using the TD and the CNT methods.

mean height of GHT constructed using the CNT method is less than that of GHT constructed using the TD method because the global number of GHT constructed using the CNT method is much higher than that using the TD method.

7.3.3 Evaluation of parallel kNN search

For the evaluation of parallel kNN search with $k = 5, 10, 15, 20, 50$ and 100 in GHT constructed using our proposed TD method, distances, comparisons and time of kNN search will be computed to reach 100 queries. To test the efficiency of the kNN search in GHT constructed using the TD method, results will be compared with those of the kNN search in GHT constructed using the CNT method.

7.3.3.1 Distances in parallel kNN search

Figure 7.7 shows the distances computed during the parallel kNN search with $k = 5, 10, 15, 20, 50$ and 100 in GHT constructed using the TD method and the CNT method. We can see that, for both datasets, the computed distances using the TD method are less than those computed using the CNT method. This is

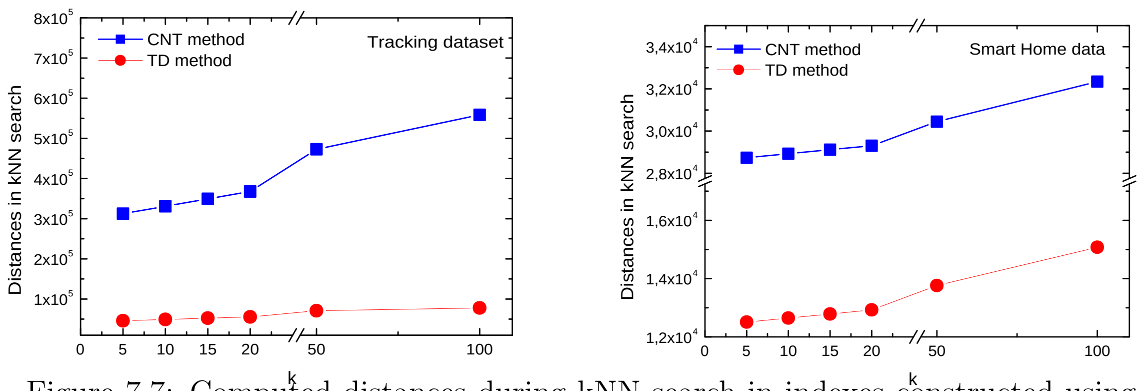


Figure 7.7: Computed distances during kNN search in indexes constructed using the TD method and CNT method.

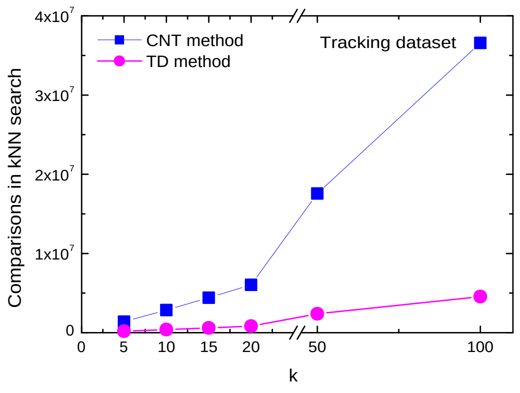
because, in the TD methods, objects in clusters are inserted in GHT of the closest clusters while in the CNT method, in spite of the closeness of clusters, a GHT is constructed for each one.

7.3.3.2 Comparisons in parallel kNN search

As awaited, the computed comparisons during the kNN search in indexes constructed using the TD method present the lowest number compared with the CNT method (Figure 7.8). This reflects the efficiency of the TD method which allows insertion of objects in some conditions related to the threshold distance. For $k > 50$, the computed comparisons of the CNT method increased considerably compared with the computed comparisons of the TD method which proves, an other time, the efficiency of the TD method for indexing continuous data flow.

7.3.3.3 Time of kNN search

In figure 7.9 is traced the time of parallel kNN search with $k = 5, 10, 15, 20, 50$ and 100 in GHT constructed using the TD method and the CNT method. We can see that the time of kNN search in the TD method GHT is better than that



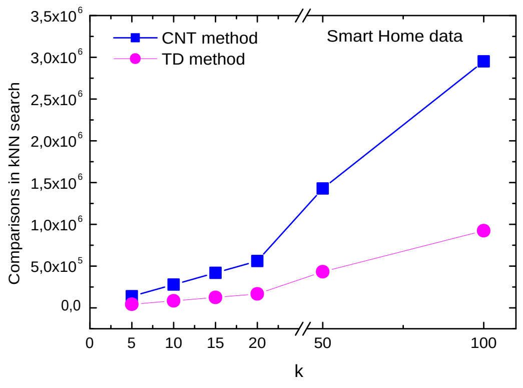
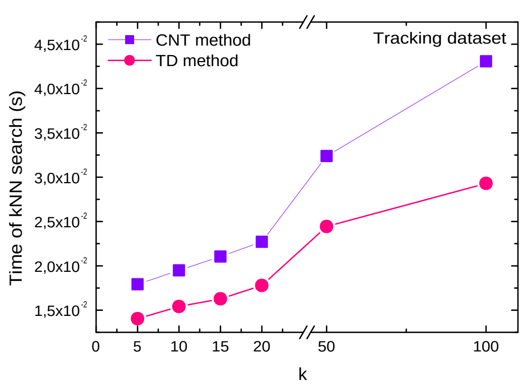


Figure 7.8: Computed comparisons during kNN search in indexes constructed using the TD method and CNT method.

in the CNT method what confirms the efficiency of the TD method for processing the continuous data flow. For $k < 30$, the time of kNN search in the TD method GHT is less than that of the CNT method by 22% for the tracking dataset and by 45% for smart home data. This difference changes for $k = 100$. The time of the kNN search for the TD method is less than that for the CNT method by 32% for the tracking dataset and by 47% for the smart home data. Even the TD method surpasses the CNT method, it proves its efficiency when comparing with the BCCF-tree [5] and the IWC-tree[5] in which, the whole dataset was indexed in one tree. The TD method is largely surpassed by the B3CF-tree in which, parallelism was used in data indexing and in kNN similarity query search.



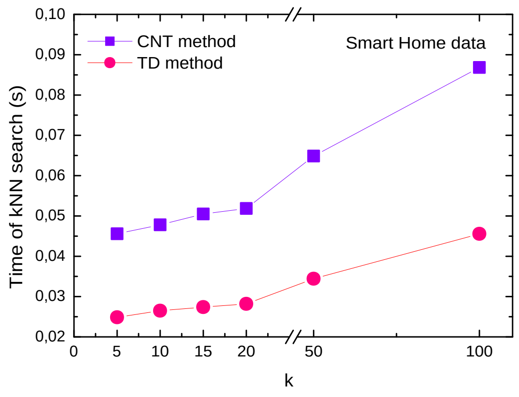


Figure 7.9: Time of kNN search in GHT constructed using the TD method and the CNT method.

It is to notice that, in order to make the comparison with the TD method, the implementation results for the BCCF-tree and the IWC-tree were obtained after computing using our own machine.

7.3.3.4 Comparison of the time of kNN search between CV and TD method

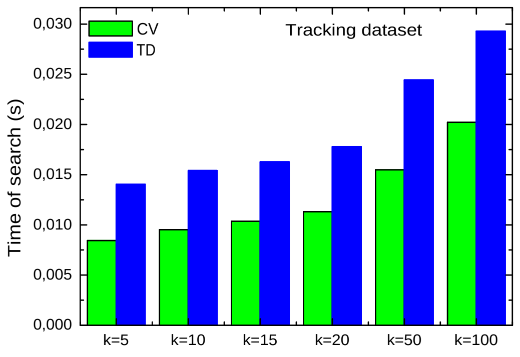


Figure 7.10: Time of search in CV and TD method.

For the tracking dataset, the only common dataset in testing CV and TD methods, the time of the kNN search for $k = 100$ was 0.029s for the TD method. The comparison of the parallel kNN search in indexes constructed using the CV method and the TD method is presented in figure 7.10 for the tracking dataset. As can be seen, the CV method surpasses the TD method. The time of kNN search for $k = 100$ is 0.02s. The parallel kNN search time presents the same evolution as a function of the parameter k for both methods. However, the TD method depend strongly on the threshold distance determination method which presents a serious limitation when indexing continuous IoT data.

7.4 Conclusion

In order to index the continuous IoT data flow, an efficient method, called threshold distance (TD) is proposed in this chapter. This method, developed in the fog-cloud architecture, is a two step process. In the first process, which takes place in the clustering fog level, the data flow is grouped into clusters by means of the DBSCAN algorithm. In the second process, in the indexing fog level, data of each cluster is inserted in an existing GHT or a new GHT is constructed after a comparison of the distance between the centers of the first cluster and the next clusters to a threshold distance value. To check the efficiency of the TD method, it was compared to an other method called the creation of a new tree (CNT). The experimental results showed that both methods are efficient compared with two other indexing methods. The experimental results showed also, that even the TD method surpassed the CNT method during the construction of GHT and the parallel kNN search, it seems insufficient in front of the CV method in term of the time of kNN query search.

8 Parallel kNN Search in QCCF-tree Nodes

8.1 Introduction

In these previous propositions, the enhancement of the efficiency of the kNN query search, in indexes, by the introduction of parallelism allowed by the use of DB-SCAN clustering algorithm, as a pre-indexing process, was evidenced. However, what about the use of parallelism in the inner of indexes? To response to this question, a new index called Quad-tree based on Containers at the Cloud-Fog computing level (QCCF-tree) inspired from the BCCF-tree [5] is proposed. It is constructed in metric space where the data is divided into four balls with four pivots. The choice of four pivots is to eliminate data overlapping and index degeneration problems. For the speed up of the kNN query search, parallelism is used in the inner of the QCCF-tree i.e. in the QCCF-tree nodes. The present chapter starts with a detailed description of the proposed approach followed by the exposition of the experimentation and the evaluation the construction and the kNN query search results of the proposed QCCF-tree by making a comparison with the results of our proposed B3CF-tree and those of some existing indexes namely BCCF-tree [5], IWC-tree [5], MX-tree [247] and BB-tree [176],[201].

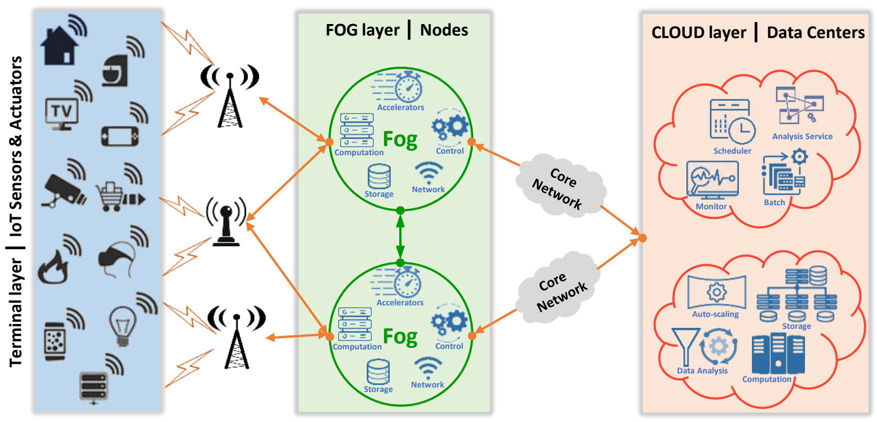


Figure 8.1: Cloud-fog computing architecture.

8.2 Proposed Approach

In spite of the efficiency of the BCCF-tree [5] in large-scale data indexing, it presents an elevated time of construction compared with other indexes such as BB-tree [176] and MX-tree [247]. In this section, we have investigated the characteristics of the cloud-fog architecture for the construction of our index called Quad-tree based on Containers at the Cloud-Fog computing level (QCCF) supposed to be the improvement of the BCCF-tree. Our system architecture, similar to that of the BCCF-tree [5], consists of three layers (Figure 8.1): the IoT devices layer (or terminal layer), the fog layer, and the cloud layer. The terminal layer sends the data generated by the interconnected IoT devices to the fog layer. The fog nodes are close to the IoT devices and have the ability to compute and store the data. In this approach, data is indexed and the QCCF-tree is constructed in the fog layer. The leaves of the nodes in the constructed QCCF-tree are stored in the cloud layer. The QCCF-tree is based on the division of the space, in the fog layer, into four non-overlapped sub-spaces (or partitions). The creation of four partitions follows a two-step process (Figure 8.2): In the first step, the space is divided into two regions, left and right, by choosing the two farthest objects as left pivot p_L and right pivot p_R. In the second step, each region is divided in turn into two partitions top and bottom. Pivots are always chosen as the farthest objects. This partitioning process results in the creation of four balls with pivots p_1, p_2, p_3 and p_4.

We define the QCCF-tree nodes N (Figure 8.2) as follow:

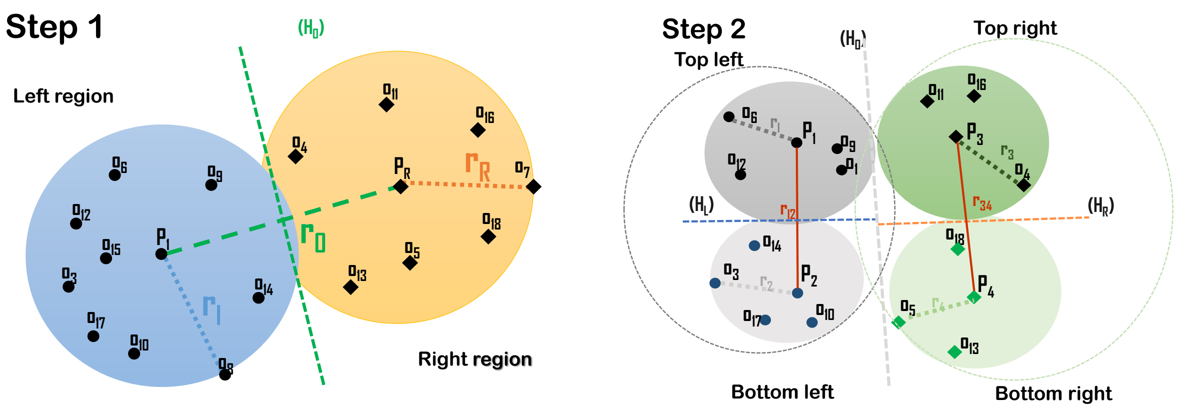


Figure 8.2: Partitioning of space in QCCF-tree.

- L leaf node- consists of a subset indexed objects:$E_L \subseteq E$ where $|E_L| \leqslant c_{max}$ the contents of the leaves partitions E.

- N Internal node is a duodecuple :

  $(p_1, p_2, p_3, p_4, r_1, r_2, r_3, r_4, r_{12}, r_{34}, N_1, N_2, N_3, N_4) \in \mathcal{O}^4 \times \mathbb{R}^6 \times \mathcal{N}^4$.

  where :

  $r_{12} = d(p_1, p_2)$ lets to define two balls $B_1(p_1, r_{12})$ and $B_2(p_2, r_{12})$, centered on p_1 and p_2 respectively and having a common radius value, large enough for the two balls to have a nonempty intersection.

  $r_{34} = d(p_3, p_4)$ lets to define two balls $B_3(p_3, r_{34})$ and $B_2(p_2, r_{34})$, centered on p_3 and p_4 respectively and having a common radius value, large enough for the two balls to have a nonempty intersection.

  (r_1, r_2, r_3, r_4) are the distances to the farthest object in the subtree rooted at that node N with respect to p_1 , p_2,p_3 and p_2 respectively.

$r_{12} = d(p_1, p_2)$

$r_{34} = d(p_3, p_4)$

(N_1, N_2, N_3, N_4) are four subtrees, such that:

$N_1 = \{o \in N : d(p_1, o) \leq d(p_2, o) \leq d(p_3, o) \leq d(p_4, o)\}$

$N_2 = \{o \in N : d(p_2, o) < d(p_1, o) < d(p_3, o) < d(p_4, o)\}$

$N_3 = \{o \in N : d(p_3, o) \leq d(p_4, o)) \leq d(p_1, o) \leq d(p_2, o)\}$

$N_4 = \{o \in N : d(p_4, o) < d(p_3, o) < d(p_1, o) < d(p_2, o)\}$

The construction of the QCCF-tree is presented in the algorithm 11.

8.2.1 QCCF-tree build

In the incremental process of the QCCF-tree construction, the insertion of objects is done from top to bottom. The formal description of the QCCF-tree construction process is presented in algorithm 1. Initially, the tree is empty and is considered as a leaf. The farthest two-pivot search algorithm is used for all objects to divide the space into two regions (left and right). After that, this algorithm is used, first, for objects in the left region to divide them into two partitions (top and bottom) with pivots p_1 and p_2 and, second, for objects in the right region to divide them into two partitions (top and bottom) with pivots p_3 and p_4. As a result, the container is divided into four non-overlapped subsets so that each element in the container belongs to its nearest pivot. This transforms the leaf into an internal node with four pivots p_1,p_2,p_3 and p_4 that create four leaf nodes (Figure 8.2).

Algorithm 11 Construction of QCCF-tree

```
Build QCCF ($S \in \mathcal{P}()) \in \mathcal{N}$
with:
$(p_1, p_2, p_3, p_4)=$ The four farthest pivots
$\triangleq \begin{cases} \perp \quad if S = \varnothing \\ (e, \perp, \perp, \perp) \quad if S = \{e\} \\ \begin{pmatrix} p_1, p_2, p_3, p_4 \\ BuildQCCF : (\{e \in S : d(p_1, e) \leqslant d(p_2, e) \leqslant d(p_3, e) \leqslant d(p_4, e)\} \setminus \{p_1\}) \\ BuildQCCF : (\{e \in S : d(p_2, e) < d(p_1, e) < d(p_3, e) < (p_4, e)\} \setminus \{p_2\}) \\ BuildQCCF : (\{e \in S : d(p_3, e) \leqslant d(p_4, e) \leqslant d(p_1, e) \leqslant d(p_2, e)\} \setminus \{p_3\}) \\ BuildQCCF : (\{e \in S : d(p_4, e) < d(p_3, e) < d(p_1, e) < d(p_2, e)\} \setminus \{p_4\}) \end{pmatrix} else \end{cases}$
```

8.2.2 Parallel kNN search in QCCF-nodes

Parallelism is used, in this approach, for the minimization of the time of the similarity search query process. Contrary to all indexing trees, parallelism is done in the QCCF-nodes level because of the presence of four pivots in each internal node, which may increase the time of the sequential kNN search process. The formal description of the kNN search in each QCCF-node is presented in algorithm 12. In each internal QCCF-node, kNN search is performed in the left region (with pivots p_1 andp_2) and the right region (with pivots p_3 and p_4) in parallel. The aim of the k-nearest neighbor search is to find the set A of objects closest to a query point q. The set A represents the fusion of the set A_{12} (in the left region) and the set A_{34} (in the right region). The kNN search algorithm starts with a query radius $rq = min(rq_{12}, rq_{34})$ where rq_{12} is the query radius in the left region, and rq_{34} is the query radius in the right region. Both radius are initialized to $+\infty$ which should lead to scanning the dataset and then decreases by traversing each node which corresponds to the distance to the k^{th} object in the ordered list A_{12} and the ordered list A_{34} respectively. Comparing $d_1(q, p_1)$ and $d_2(q, p_2)$ with rq_{12} and $d_3(q, p_3)$ and $d_4(q, p_4)$ with rq_{34} indicates the descent of the query point in the index. The leaf nodes contain a subset of the indexed data with a maximum

Algorithm 12 Parallel kNN search in QCCF-nodes.

$$\text{kNN-QCCF}\left(\begin{array}{l} N \in \mathcal{N}, \\ q \in \mathbb{R}^n, \\ k \in \mathbb{N}^*, \\ d : \mathcal{O} \times \mathcal{O} \to \mathbb{R}^+, \\ r_q \in \mathbb{R}^+ = +\infty, \\ A \in (\mathbb{R}^+ \times \mathcal{O})^{\mathbb{N}} = \emptyset \end{array}\right) \in (\mathbb{R}^+ \times \mathcal{O})^{\mathbb{N}}$$

with :

- $A_{12} = kNN{-}QCCF(L, q, k, d, rq_{12}, k{-}insert(A_{12}, ((d(p_1, q), p), d(p_2, q), p)))$
- $A_{34}{=}kNN - QCCF(R, q, k, d, rq_{34}, k - insert(A_{34}, ((d(p_3, q), p), d(p_4, q), p)))$
- $rq_{12} = max\{d : (d, o) \in A_{12}\}$ if $|A_{12}| = k$
- $rq_{34} = max\{d : (d, o) \in A_{34}\}$ if $|A_{34}| = k$
- $rq = min(rq_{12}, rq_{34})$
- $A = A_{12} \cup A_{23}$

$$\triangleq \begin{cases} A, \text{if}(N = \perp) \\ A_{12}, \text{if}(N = (p, r, L, R) \wedge d(q, p_1) < rq_{12} \wedge d(q, p_2) < rq_{12}) \parallel \\ A_{34}, \text{if}(N = (p, r, L, R) \wedge d(q, p_3) < rq_{34} \wedge d(q, p_4) < rq_{34}) \end{cases}$$

cardinal c_{max}. To find the k nearest neighbors of a leaf, we simply sort the indexed data according to their increasing distances to the query q. As a result of the search, the first k sorted objects in the list A are returned.

8.3 Simulation and Results

To perform our experiments, two datasets of different sizes and dimensions are used.

1. Geographical coordinate database: a real dataset of 988 2D vectors. It contains BD-L-TC topographic data of selected locations and places [270].
2. Tracking of a moving object: a real dataset of 62702 20D vectors. It represents the results of a random simulation of tracking a moving object using wireless cameras [5].

The experiments were performed using the Python programming language installed on an Intel ® CoreTM i7-8550u CPU, 1.80 GHz*8 processor with a 16 Gb RAM, a 256 Gb SDD ROM and 64-bit Linux operating system (Ubuntu). The aim of the experiments is to analyze the effectiveness of the proposed QCCF-tree index construction and query response by comparing our results to those obtained with the following index structures:

- BCCF-tree (Binary tree based on containers at the cloud-fog computing level) [5]: this index is based on recursive space partitioning using k-means clustering algorithm to efficiently separate the space into two subspaces.
- IWC-tree (Indexing tree without containers) [5]: The comparison of our results with those of this index can show the effectiveness of using containers in binary trees.
- MX-tree [247]: The comparison of our results with those of this index can highlight the difference between hyperplane and ball partitioning in metric space.
- BB-tree (Bubble Buckets tree) [176]: A comparison with our proposed index will show the difference between the metric space structure and the multidimensional space structure.

8.3.1 Evaluation of the QCCF-tree construction

In this section, the evaluation of the construction of the QCCF-tree, wil be done by evaluating the number of computed distances(Figure 8.3), the number of comparisons(Figure 8.4), and the construction time (Figure 8.5). The size of the container in the QCCF-tree is set as $c_{max} = \sqrt{n}$.

8.3.1.1 Number of calculated distances

As can be seen, in figure 8.3, the number of calculated distances varies from a data to another. However, for both used data, the minimum number of distances is calculated for the BB-tree while the maximum number of distances is calculated for the IWC-tree. This is awaiting from these two indexes since the BB-tree is based on multidimensional space partitioning without distances calculation and, in the IWC-tree, the indexing is done for the whole objects in the data. The proposed QCCF-tree exhibits the lowest number of distances compared with the BCCF-tree and the IWC-tree, for tracking dataest. For the geographical coordinates data, the number of the calculated distances in the QCCF-tree is close to that of the MX-tree. Partially indexing of data in containers diminished considerably the number of distances when compared with the IWC-tree. For the geographical coordinates data, the number of distances computed in the BCCF-tree is comparable to that computed in the IWC-tree and this may be due to the use of k-means algorithm for the determination of the two pivots in the BCCF-tree.

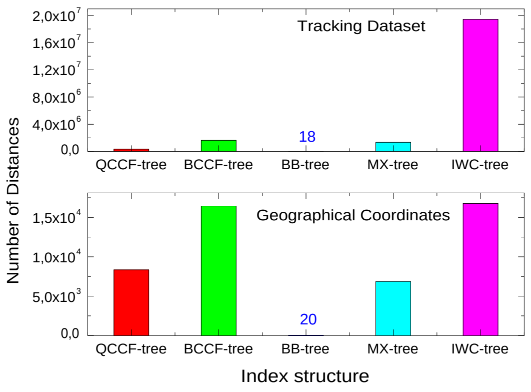


Figure 8.3: Number of distances of QCCF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree.

8.3.1.2 Number of comparisons

The number of comparisons is presented, in figure 8.4, for the selected indexes. One can see that the number of comparisons calculated for our proposed QCCF-tree is the lowest compared with the other indexes which indicates its efficiency. Indeed, the division of data into four subsets, in the container, results in the creation of subsets containing closest objects which induces the reduction of the number of comparisons. Contrary to the number of distances, the BB-tree exhibits the elevated number of comparison.

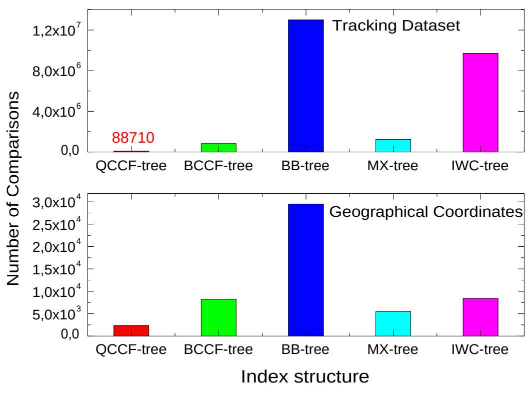


Figure 8.4: Number of comparisons calculated of QCCF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree.

8.3.1.3 Construction time

The variation of the construction time of the chosen index structures is presented in figure 6.6. For the tracking dataset, the construction time of our proposed QCCF-tree is lower than that of the BCCF-tree and the IWC-tree and, it is comparable to those of the BB-tree and the MX-tree. For the geographical coordinates data, the construction time of the proposed QCCF-tree is close to that of the BCCF-tree, where the difference of the time is about 0.015 second, and is greater than those of the BB-tree, the MX-tree and the IWC-tree with a difference of time around

0.04 second.

According to the results of the number of distances, the number of comparisons and the low difference in the construction time, the proposed QCCF-tree could be considered as a competitive structure for in-fog IoT data indexing.

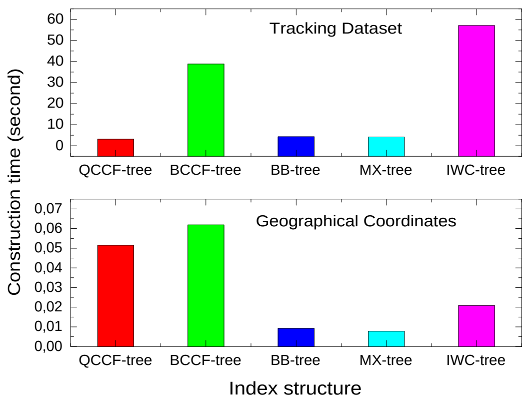


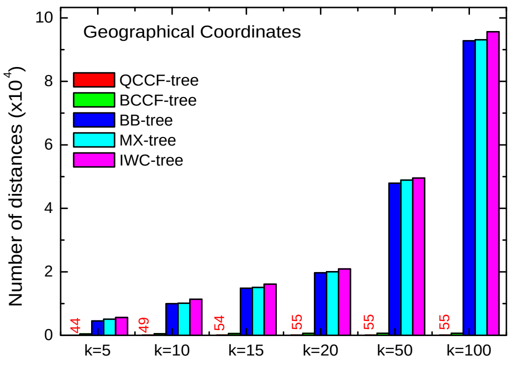
Figure 8.5: Construction time of QCCF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree.

8.3.2 Evaluation of the in-node parallel kNN search

For the evaluation of the in-node parallel kNN search of similarity query, the number of distances, the number of comparisons and the search time , for both used datasets, are taken as the average of 100 queries. The variation of these three characteristics as a function of the parameter k, where $k = 5, 10, 15, 20, 50$ and 100, are compared with the results of the BCCF-tree, the BB-tree, the MX-tree and the IWC-tree .

8.3.2.1 Number of calculated distances

For both used datasets, the calculated number of distances in the proposed QCCF-tree is less than that of the other structures (Figure 8.6) which reflects the efficiency of the use of four balls with four pivots for data partitioning and the parallelism when browsing the index during the similarity query search. For the geographical coordinates data, the number of distances, calculated in the QCCF-tree, increases from 44, for $k = 5$, and stabilises at 55 from $k = 20$. For the tracking dataset, the number of distances, calculated for the QCCF-tree, increased without stabilizing. However, the ratio between the number of distances for $k = 5$ and that for $k = 100$, which is 78%, indicates that the number of distances nearly invariant as a function of the parameter k.



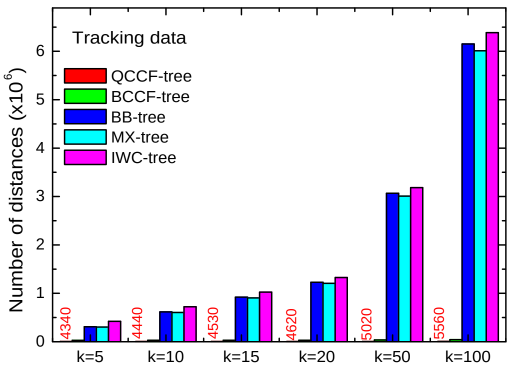


Figure 8.6: Number of distances calculated for the kNN search in QCCF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree.

8.3.2.2 Number of calculated comparisons

In figure 8.7, is presented the calculated number of comparisons as a function of the parameter k. Like for the number of distances, the QCCF-tree exhibits the lowest number of comparisons compared with the other indexes. This also confirms the efficiency of our proposed index in the similarity query search. For the geographical coordinates, the calculated number of comparisons, in the QCCF-tree, increases from 84, for $k = 5$ and stabilises at 139 beyond $k = 15$. For the tracking dataset,the calculated number of comparisons, in the QCCF-tree, increases by a magnitude of

10 from $k = 5$ (20.239×10^3) to $k = 100$ (26.4251×10^4). For the same dataset, the number of comparisons, in the QCCF-tree represents 15% of the number of comparisons, in the BCCF-tree, for $k = 5$ and 13% for $k = 100$.

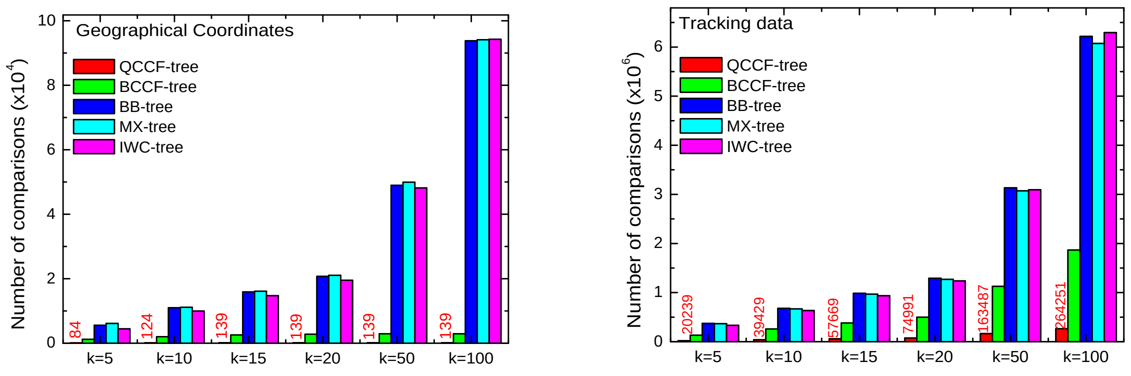


Figure 8.7: Number of comparisons calculated for the kNN search in QCCF-tree, BCCF-tree,BB-tree, MX-tree and IWC-tree.

8.3.2.3 Time of search

The efficiency of an index could be evaluated from the data retrieve time. The in-node parallel kNN search time in the proposed QCCF-tree is plotted, in figure 8.8, with the search time of the BCCF-tree, the BB-tree, the MX-tree and the IWC-tree as a function of the parameter k.

As can be seen, in figure 8.8, the search time in the QCCF-tree has the lowest value compared with the other indexes. This was awaited after the evaluation of the number of distances and the number of comparisons where the in-node parallel search efficiency was evidenced. For the geographical coordinates data, the kNN search time, in the QCCF-tree, is mainly invariant, as a function of the parameter k, with a mean value of 0.0013s which is less than the search time, in the BCCF-tree (0.0025s for $k = 5$).

For the tracking dataset, the kNN search time, in the QCCF-tree, is also invarient as a function of the parameter k.

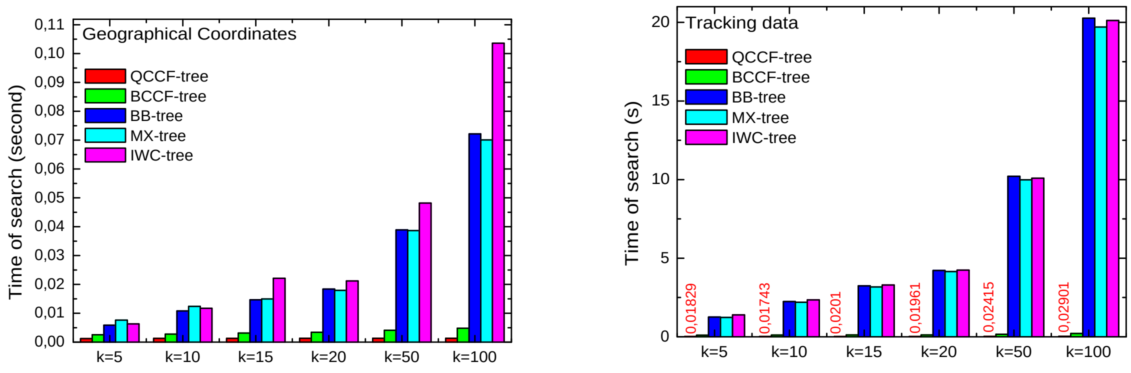


Figure 8.8: Time of kNN search in QCCF-tree, BCCF-tree, BB-tree, MX-tree and IWC-tree.

Its mean time of 0.021s is less than the search time, in the BCCF-tree, for $k = 5$ which is equal to 0.1s. Our results are also comparable to those in literature. In [237], for $k = 50$, the query time is 0.1s on Foursquare dataset while in [209], the execution time for spatial range query on R*-tree on spark is 0.02s. In [263], with a number of workers of 32 and for $k = 4$, the query cost 2.7s for CoPHIR dataset.

8.3.3 Comparison between B3CF-tree and QCCF-tree

Face to the above interesting results, the QCCF-tree could be considered as the improvement of the BCCF-tree especially during the combination of parallelism with the kNN search method. However, a comparison with our next proposed index (B3CF-tree), in which parallelism is used during indexes construction and the kNN query search, must be done to find whether the index that could be considered as an efficient alternative for IoT data indexing, storing and searching.

According to the above experimental results, our metric space proposed approach proved its efficiency regarding the use of parallelism during both the B3CF-tree construction and the kNN query search. However, a confrontation to the QCCF-tree must be done in order to find the best alternative for big IoT data indexing and retrieving. The kNN search time, with $k = 5, 10, 15, 20, 50$ and 100, in the the B3CF-tree is presented with the kNN search time in QCCF-tree nodes in figure 8.9 for the geographical coordinates and tracking datasets. As can be seen, the B3CF-tree presents much better results compared with the QCCF-tree.

Even for the construction time, figure 8.10 shows that the results of the B3CF-tree are also much better than those of the QCCF-tree for both datasets which, without a doubt, make of the B3CF-tree the efficient alternative for IoT data indexing and queries retrieving.

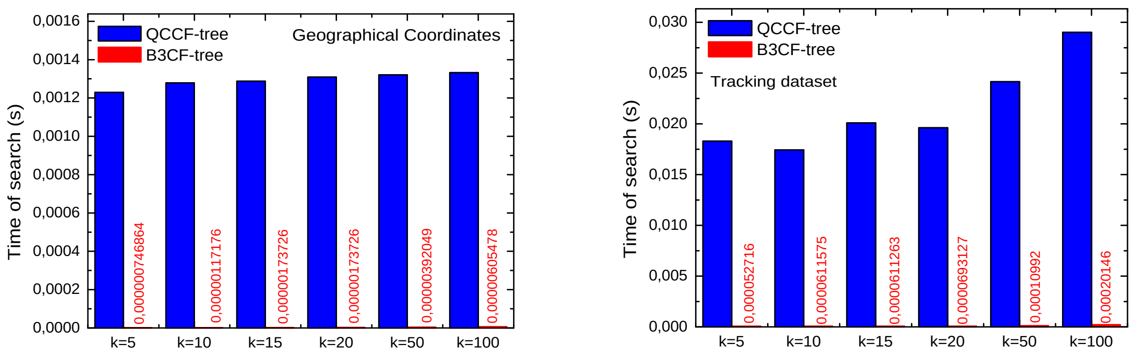


Figure 8.9: Time of kNN search in QCCF-tree and B3CF-tree.

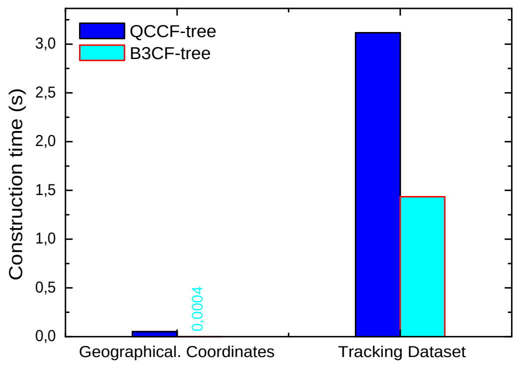


Figure 8.10: Construction time in QCCF-tree and B3CF-tree.

However, despite its efficiency, the B3CF-tree faced a limitation that is the query search cost. Indeed, the use of parallelism makes the kNN query search simultaneous in all fogs, which will multiply the consumed energy taking into consideration that the search result will be finely send from only one index in one fog. The sequential kNN search does not consume energy compared with parallel kNN search. However, it presents a latency problem because if the query is not found in one fog, the kNN search will be done in the next fog and so on.

8.4 Conclusion

In this chapter, the use of four balls with four pivots partitioning was done in order to overcome the problem of the efficiency of indexing, storing and retrieving IoT big data. This is because the dividing of the exponentially-grown data into subsets using balls, with one or two pivots, induced the degeneration of the index due to the inherent inadequacy of space partitioning. Our proposed index structure, called QCCF-tree, exhibited interesting and competitive experimental results either in the construction or in the similar query search using parallelism when browsing the index nodes. Indeed, the comparison of the index construction evaluation and the similarity search results of the QCCF-tree with those of the BCCF-tree [5], IWC-tree [5], MX-tree [247] and BB-tree [176],[201] showed that the QCCF-tree surpassed them largely. However, when comparing these results with those of the B3CF-tree, the QCCF-tree exhibited a remarkable insufficiency in both index construction and similarity query search.

Conclusions

This thesis work represents our contribution in the similarity queries search in metric space in IoT systems. The kNN method was used for similarity queries search in proposed indexes developed, in metric space, using the fog-cloud architecture which contains a terminal layer, a fog layer and a cloud layer. The indexing process is shifted from the cloud to the fog nodes to get the data near the indexing structure and thus, reduce the network traffic congestion significantly. Moreover, each fog node creates its unique indexing structure, allowing not only parallelism during trees construction, but also parallelism in the search process by launching the same query simultaneously on all fog nodes. In the first proposed approach, called the B3CF-tree (Binary tree based on Containers at the Cloud-Clusters Fog computing level), each fog node is divided into clustering fog level and indexing fog level. In the clustering fog level, IoT data sent from the terminal layer is partitioned into homogeneous groups, or clusters, in terms of type and dimension using the DBSCAN algorithm. The aim of the clustering process was to generate a balanced trees with a reduced degree of overlapping between leaves. In the indexing fog level, objects in each cluster are indexed, in parallel, in B3CF-trees. The data partitioning, using the DBSCAN algorithm, allowed parallelism, not only when indexing data of the resulting clusters, but also, when using the kNN method for the similarity queries search. The experimental results, obtained using one data stream, showed that the proposed B3CF-tree outperforms indexes in literature, such as BCCF-tree, IWC-tree and BB-tree, in terms of indexes construction, indexes quality and the similarity query search using the kNN method.

For indexing continuous data stream generated from IoT devices, two other approaches were proposed: the Coefficient of Variation (CV) method and the Threshold Distance (TD) method. For both proposed approaches, the kNN search method was combined with parallelism when searching the similarity queries.

In the CV method approach, the fog layer is divided into three levels: the clustering level, the clusters processing level and the indexing level. In the clustering fog level, the first data stream is grouped into clusters using DBSCAN algorithm which are stored in the clusters processing fog level while their corresponding BH-trees are directly constructed, in parallel, in the indexing fog level. After the clustering of the arrival data stream, in the clustering level, the coefficient of variation (CV) of the union of of each arrival cluster with the a copy of first clusters is calculated and, according to the CV value, objects of the arrival cluster are inserted into an existing BH-tree or a new BH-tree is constructed. To test the efficiency of the proposed CV method, two other scenarios were proposed for comparison. In the first scenario, called the Creation of a New Index (CNI) method, for each arrival cluster, a BH-tree is constructed. In the second scenario, called the Insertion in an Existing Index (IEI) method, the objects of each arrival cluster are inserted in an existing BH-tree corresponding to the closest existing cluster. From the evaluation of BH-trees construction, the IEI method surpassed the CV and CNI methods. Parameters of the CV method are always located between those of the CNI and the IEI methods. For the parallel kNN query search, the three proposed methods where efficient compared with other methods from literature. The comparison of the proposed Cv method with the proposed scenarios showed that the CV method is more efficient than the CNI and the IEI methods in terms of the parallel kNN similarity query search and the energy consumption.

In the TD method approach, the for layer, as for the B3CF-tree structure, is divided into a clustering level and an indexing level. In the clustering level, as for the CV method, the first data stream is grouped into clusters by means of the DBSCAN algorithm. Centers clusters are also determined. In the indexing level, Generalised Hyper-plane Trees (GHT) are constructed, in parallel, for each first cluster. The center of each first cluster is taken as a representative of the corresponding GHT. After the clustering and the determination of the clusters centers of the arrival data stream, objects in each arrival cluster are inserted in an existing GHT or a new GHT is constructed basing on the comparison of the distances between the arrival cluster center and the existing GHT representatives to a threshold distance. To check the efficiency of the TD method, it was compared to a proposed scenario called the Creation of a New Tree (CNT). The experimental results showed that both methods are efficient compared with other indexing methods from literature. The experimental results showed also, that the TD method surpassed the CNT method not only during the construction of GHT but also during the parallel kNN search of similarity queries. However, the TD method presented some weakness when compared with the CV method.

In last proposition of this work, the fog node was not divided in the cloud-fog architecture. In the fog node, the proposed QCCF-tree (Quad-tree based on Containers at the Cloud-Fog computing level) is based on the use of four balls with four pivots partitioning in metric space. This approach was proposed in order to overcome the problem of the efficiency of indexing, storing and retrieving of big IoT data. The proposed QCCF-tree exhibited interesting and competitive experimental results either in the index construction or in the parallel kNN similarity query search in the inner of the QCCF-tree i.e. in the QCCF-tree nodes. The confrontation of the experimental results to those of BCCF-tree, BB-tree, MX-tree and IWC-tree showed that the performances of the proposed QCCF-tree surpasses their largely whether it be in the index construction or in the similarity query search. The evaluation and the comparison results between QCCF-tree and B3CF-tree clearly showed that the efficiency of parallel similarity query search and the quality of B3CF-tree indexes exceeded those of the QCCF-tree. Indeed, the introduction of parallelism allowed by the DBSCAN clustering improved the construction characteristics of the B3CF-tree and also, significantly accelerated the kNN similarity queries search.

As a future work, we will focus on the implementation of the algorithm in real IoT networks and testing real data from real situations. Despite the evidenced effect of parallelism in improving the indexing and the retrieving processes of big IoT data in term of time, it presents the disadvantage of cost i.e. the energy consumption. Indeed, the use of parallelism, especially when searching similarity queries, induced the exploration of all machines simultaneously while the answer is sent from only on machine. The proposition of an alternative of parallelism that reduces the energy consumption and guards the same efficiency of the proposed methods in this thesis work will be also considered.

References

[1] V. Upadrista, “The iot standards reference model,” in IoT Standards with Blockchain, pp. 61–86, Springer, 2021. 1

[2] M. Taneja and A. Davy, “Resource aware placement of iot application modules in fog-cloud computing paradigm,” in 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 1222–1228, IEEE, 2017. 1

[3] M. Wang and Q. Zhang, “Optimized data storage algorithm of iot based on cloud computing in distributed system,” Computer Communications, vol. 157, pp. 124–131, 2020. 1

[4] M. Zhou, J. Li, and J. Ye, “Design of big data compatible storage system based on cloud computing environment,” in 2020 39th Chinese Control Conference (CCC), pp. 3158–3161, IEEE, 2020. 1

[5] A.-E. Benrazek, Z. Kouahla, B. Farou, M. A. Ferrag, H. Seridi, and M. Kurulay, “An efficient indexing for internet of things massive data based on cloud-fog computing,” Transactions on emerging telecommunications technologies, vol. 31, no. 3, p. e3868, 2020. 1, 119, 120, 128, 132, 136, 137, 143, 159, 160, 173, 190, 197, 204, 207, 208, 212, 213, 222

[6] D. Miao, L. Liu, R. Xu, J. Panneerselvam, Y. Wu, and W. Xu, “An effi-

cient indexing model for the fog layer of industrial internet of things," IEEE Transactions on Industrial Informatics, vol. 14, no. 10, pp. 4487–4496, 2018. 1, 119, 128

[7] C. Böhm, S. Berchtold, H.-P. Kriegel, and U. Michel, "Multidimensional index structures in relational databases," Journal of Intelligent Information Systems, vol. 15, no. 1, pp. 51–70, 2000. 2

[8] V. Gaede and O. Günther, "Multidimensional access methods," ACM Computing Surveys (CSUR), vol. 30, no. 2, pp. 170–231, 1998. 2

[9] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M. Ferrag, N. Choudhury, and V. Kumar, "Security and privacy in fog computing: Challenges," IEEE Access, vol. 5, pp. 19293–19304, 2017. 2

[10] D. Puschmann, P. Barnaghi, and R. Tafazolli, "Adaptive clustering for dynamic iot data streams," IEEE Internet of Things Journal, vol. 4, no. 1, pp. 64–74, 2016. 3, 9, 52, 57, 65, 67, 71

[11] G. Fortino and P. Trunfio, Internet of things based on smart objects: Technology, middleware and applications. Springer, 2014. 9

[12] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in Proceedings of the 1984 ACM SIGMOD international conference on Management of data, pp. 47–57, 1984. 9, 81, 85, 86, 92, 124

[13] C. Yang, D. Deng, S. Shang, and L. Shao, "Efficient locality-sensitive hashing over high-dimensional data streams," in 2020 IEEE 36th International Conference on Data Engineering (ICDE), pp. 1986–1989, IEEE, 2020. 9, 75, 90, 121, 125

[14] Z. Chen, B. Yao, Z.-J. Wang, W. Zhang, K. Zheng, P. Kalnis, and F. Tang, “Itiss: an efficient framework for querying big temporal data,” GeoInformatica, vol. 24, no. 1, pp. 27–59, 2020. 9, 91, 125

[15] P. Zezula, G. Amato, V. Dohnal, and M. Batko, Similarity search: the metric space approach, vol. 32. Springer Science & Business Media, 2006. 9, 12, 16, 17, 18, 94, 98

[16] R. Mao, H. Xu, W. Wu, J. Li, Y. Li, and M. Lu, “Overcoming the challenge of variety: big data abstraction, the next evolution of data management for aal communication systems,” IEEE Communications Magazine, vol. 53, no. 1, pp. 42–47, 2015. 9

[17] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín, “Searching in metric spaces,” ACM computing surveys (CSUR), vol. 33, no. 3, pp. 273–321, 2001. 9, 10, 101

[18] Z. Kouahla, Indexation dans les espaces métriques Index arborescent et parallélisation. PhD thesis, Université de Nantes, 2013. 11, 12, 14, 15, 16, 18, 110

[19] M. L. Hetland, “The basic principles of metric indexing,” in Swarm intelligence for multi-objective problems in data mining, pp. 199–232, Springer, 2009. 15

[20] C. Zavazava, “Itu work on internet of things,” in Presentation at ICTP workshop, 2015. 20, 23

[21] V. Sharma and R. Tiwari, “A review paper on “iot” & it’s smart applications,” International Journal of Science, Engineering and Technology Research (IJSETR), vol. 5, no. 2, pp. 472–476, 2016. 20, 23, 27

[22] D. Happ, "Cloud and fog computing in the internet of things," Internet of Things A to Z: Technologies and Applications; Wiley Online Library: New York, NY, USA, pp. 113–134, 2018. 20, 130

[23] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," Future generation computer systems, vol. 29, no. 7, pp. 1645–1660, 2013. 21, 22, 27, 28, 29

[24] M. Weiser, R. Gold, and J. S. Brown, "The origins of ubiquitous computing research at parc in the late 1980s," IBM systems journal, vol. 38, no. 4, pp. 693–696, 1999. 21

[25] Y. Rogers, "Moving on from weiser's vision of calm computing: Engaging ubicomp experiences," in International conference on Ubiquitous computing, pp. 404–421, Springer, 2006. 21

[26] R. Caceres and A. Friday, "Ubicomp systems at 20: Progress, opportunities, and challenges," IEEE Pervasive Computing, vol. 11, no. 1, pp. 14–21, 2011. 21

[27] K. Ashton et al., "That 'internet of things' thing," RFID journal, vol. 22, no. 7, pp. 97–114, 2009. 21

[28] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, "Vision and challenges for realising the internet of things," Cluster of European research projects on the internet of things, European Commision, vol. 3, no. 3, pp. 34–36, 2010. 21, 22

[29] R. Van Kranenburg, The Internet of Things: A critique of ambient technology and the all-seeing network of RFID. Institute of Network Cultures, 2008. 22

[30] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," Computer networks, vol. 54, no. 15, pp. 2787–2805, 2010. 22, 26

[31] M. P. A. Hukeri, M. Ghewari, et al., "Review paper on iot based technology," International Research Journal of Engineering and Technology, vol. 4, no. 01, 2017. 23

[32] P. P. Ray, "A survey on internet of things architectures," Journal of King Saud University-Computer and Information Sciences, vol. 30, no. 3, pp. 291–319, 2018. 23, 24

[33] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," IEEE communications surveys & tutorials, vol. 17, no. 4, pp. 2347–2376, 2015. 24, 26, 30, 31

[34] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet: the internet of things architecture, possible applications and key challenges," in 2012 10th international conference on frontiers of information technology, pp. 257–260, IEEE, 2012. 24, 25, 26

[35] Z. Yang, Y. Yue, Y. Yang, Y. Peng, X. Wang, and W. Liu, "Study and application on the architecture and key technologies for iot," in 2011 International Conference on Multimedia Technology, pp. 747–751, IEEE, 2011. 24, 26

[36] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du, "Research on the architecture of internet of things," in 2010 3rd international conference on advanced computer theory and engineering (ICACTE), vol. 5, pp. V5–484, IEEE, 2010. 24, 26

[37] D. Uckelmann, M. Harrison, and F. Michahelles, "An architectural approach

towards the future internet of things," in Architecting the internet of things, pp. 1–24, Springer, 2011. 25

[38] B. Paul, "Internet of things (iot), three-layer architecture, security issues and counter measures," in ICT Analysis and Applications, pp. 23–34, Springer, 2022. 25

[39] A.-E. BENRAZEK, Internet of Things: Analysis of suspicious behaviour in a surveillance camera network. PhD thesis, 2021. 28, 130, 139

[40] H.-E. Lin, R. Zito, M. Taylor, et al., "A review of travel-time prediction in transport and logistics," in Proceedings of the Eastern Asia Society for transportation studies, vol. 5, pp. 1433–1448, Bangkok, Thailand, 2005. 29

[41] M. Navajo, I. Ballesteros, S. D'Elia, A. Sassen, M. Goyet, J. Santaella, et al., "Draft report of the task force on interdisciplinary research activities applicable to the future internet," European Union Task Force Report, 2010. 29

[42] D. Tang, Event detection in sensor networks. PhD thesis, The George Washington University, 2009. 29

[43] A. Juels, "Rfid security and privacy: A research survey," IEEE journal on selected areas in communications, vol. 24, no. 2, pp. 381–394, 2006. 29

[44] S. Wang, Z. Zhang, Z. Ye, X. Wang, X. Lin, and S. Chen, "Application of environmental internet of things on water quality management of urban scenic river," International Journal of Sustainable Development & World Ecology, vol. 20, no. 3, pp. 216–222, 2013. 30

[45] J. Kempf, J. Arkko, N. Beheshti, and K. Yedavalli, "Thoughts on reliability in the internet of things," in Interconnecting smart objects with the Internet

workshop, vol. 1, pp. 1–4, Internet Architecture Board Boston, MA, USA, 2011. 30

[46] A. Dunkels, J. Eriksson, and N. Tsiftes, “Low-power interoperability for the ipv6-based internet of things,” in Proceedings of the 10th Scandinavian Workshop on Wireless Ad-Hoc Networks (ADHOC’11), Stockholm, Sweden, pp. 10–11, 2011. 32

[47] P. Mell, T. Grance, et al., “The nist definition of cloud computing,” 2011. 32, 33, 34, 36

[48] P. Srivastava and R. Khan, “A review paper on cloud computing,” International Journal of Advanced Research in Computer Science and Software Engineering, vol. 8, no. 6, pp. 17–20, 2018. 34

[49] M. Aazam, I. Khan, A. A. Alsaffar, and E.-N. Huh, “Cloud of things: Integrating internet of things and cloud computing and the issues involved,” in Proceedings of 2014 11th International Bhurban Conference on Applied Sciences & Technology (IBCAST) Islamabad, Pakistan, 14th-18th January, 2014, pp. 414–419, IEEE, 2014. 35, 38, 46

[50] N. Khan, N. Ahmad, T. Herawan, and Z. Inayat, “Cloud computing: Locally sub-clouds instead of globally one cloud,” International Journal of Cloud Applications and Computing (IJCAC), vol. 2, no. 3, pp. 68–85, 2012. 37

[51] R. Gravina, P. Alinia, H. Ghasemzadeh, and G. Fortino, “Multi-sensor fusion in body sensor networks: State-of-the-art and research challenges,” Information Fusion, vol. 35, pp. 68–80, 2017. 38

[52] P. Tan, H. Wu, P. Li, and H. Xu, “Teaching management system with applications of rfid and iot technology,” Education Sciences, vol. 8, no. 1, p. 26, 2018. 38

[53] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M. A. Ferrag, N. Choudhury, and V. Kumar, "Security and privacy in fog computing: Challenges," IEEE Access, vol. 5, pp. 19293–19304, 2017. 38, 46, 163, 164

[54] Y. Shi, S. Abhilash, and K. Hwang, "Cloudlet mesh for securing mobile clouds from intrusions and network attacks," in 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, pp. 109–118, IEEE, 2015. 38

[55] S. Shin and G. Gu, "Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in 2012 20th IEEE international conference on network protocols (ICNP), pp. 1–6, IEEE, 2012. 38

[56] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," Future Generation Computer Systems, vol. 78, pp. 680–698, 2018. 39

[57] X. Li, X. Jiang, P. Garraghan, and Z. Wu, "Holistic energy and failure aware workload scheduling in cloud datacenters," Future Generation Computer Systems, vol. 78, pp. 887–900, 2018. 39

[58] S. Tuli, N. Basumatary, S. S. Gill, M. Kahani, R. C. Arya, G. S. Wander, and R. Buyya, "Healthfog: An ensemble deep learning based smart healthcare system for automatic diagnosis of heart diseases in integrated iot and fog computing environments," Future Generation Computer Systems, vol. 104, pp. 187–200, 2020. 39

[59] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. Netto, et al., "A manifesto for

future generation cloud computing: Research directions for the next decade," ACM computing surveys (CSUR), vol. 51, no. 5, pp. 1–38, 2018. 39

[60] Y.-K. Chen, "Challenges and opportunities of internet of things," in 17th Asia and South Pacific design automation conference, pp. 383–388, IEEE, 2012. 39

[61] M. M. Sadeeq, N. M. Abdulkareem, S. R. Zeebaree, D. M. Ahmed, A. S. Sami, and R. R. Zebari, "Iot and cloud computing issues, challenges and opportunities: A review," Qubahan Academic Journal, vol. 1, no. 2, pp. 1–7, 2021. 39

[62] S. Singh and I. Chana, "A survey on resource scheduling in cloud computing: Issues and challenges," Journal of grid computing, vol. 14, no. 2, pp. 217–264, 2016. 40

[63] S. S. Gill, P. Garraghan, and R. Buyya, "Router: Fog enabled cloud based intelligent resource management approach for smart home iot devices," Journal of Systems and Software, vol. 154, pp. 125–138, 2019. 40

[64] S. S. Gill, P. Garraghan, V. Stankovski, G. Casale, R. K. Thulasiram, S. K. Ghosh, K. Ramamohanarao, and R. Buyya, "Holistic resource management for sustainable and reliable cloud computing: An innovative solution to global challenge," Journal of Systems and Software, vol. 155, pp. 104–129, 2019. 40

[65] S. Singh and I. Chana, "Qos-aware autonomic resource management in cloud computing: a systematic review," ACM Computing Surveys (CSUR), vol. 48, no. 3, pp. 1–46, 2015. 40

[66] S. S. Gill, S. Tuli, M. Xu, I. Singh, K. V. Singh, D. Lindsay, S. Tuli, D. Smirnova, M. Singh, U. Jain, et al., "Transformative effects of iot,

blockchain and artificial intelligence on cloud computing: Evolution, vision, trends and open challenges," Internet of Things, vol. 8, p. 100118, 2019. 40

[67] M. Iorga, L. Feldman, R. Barton, M. Martin, N. Goren, and C. Mahmoudi, "The nist definition of fog computing," tech. rep., National Institute of Standards and Technology, 2017. 41, 42, 44

[68] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," ACM SIGCOMM computer communication Review, vol. 44, no. 5, pp. 27–32, 2014. 41

[69] B. Di Martino, M. Rak, M. Ficco, A. Esposito, S. A. Maisto, and S. Nacchia, "Internet of things reference architectures, security and interoperability: A survey," Internet of Things, vol. 1, pp. 99–112, 2018. 41

[70] M. R. Anawar, S. Wang, M. Azam Zia, A. K. Jadoon, U. Akram, and S. Raza, "Fog computing: An overview of big iot data analytics," Wireless Communications and Mobile Computing, vol. 2018, 2018. 42, 49, 50

[71] P. More, "Review of implementing fog computing," International Journal of Research in Engineering and Technology, vol. 4, no. 06, pp. 335–338, 2015. 43

[72] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. S. Goren, C. Mahmoudi, et al., "Fog computing conceptual model," 2018. 44, 45

[73] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," IEEE internet of things journal, vol. 4, no. 5, pp. 1125–1142, 2017. 46

[74] P. Habibi, M. Farhoudi, S. Kazemian, S. Khorsandi, and A. Leon-Garcia, "Fog computing: a comprehensive architectural survey," IEEE Access, vol. 8, pp. 69105–69133, 2020. 46

[75] J. Zhu, D. S. Chan, M. S. Prabhu, P. Natarajan, H. Hu, and F. Bonomi, "Improving web sites performance using edge servers in fog computing architecture," in 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, pp. 320–323, IEEE, 2013. 46

[76] B. Tang, Z. Chen, G. Hefferman, T. Wei, H. He, and Q. Yang, "A hierarchical distributed fog computing architecture for big data analysis in smart cities," in Proceedings of the ASE BigData & SocialInformatics 2015, pp. 1–6, 2015. 46

[77] H. R. Arkian, A. Diyanat, and A. Pourkhalili, "Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications," Journal of Network and Computer Applications, vol. 82, pp. 152–165, 2017. 46

[78] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," in Internet of things, pp. 61–75, Elsevier, 2016. 46

[79] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, "Fog computing: Survey of trends, architectures, requirements, and research directions," IEEE access, vol. 6, pp. 47980–48009, 2018. 46

[80] O. Consortium and A. Working, "Openfog reference architecture for fog computing," J.Netw. Comput, p. 1–162, 2017. 46

[81] Z. Á. Mann, "Notions of architecture in fog computing," Computing, vol. 103, no. 1, pp. 51–73, 2021. 47

[82] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: architecture, key technologies, applications and open issues," Journal of network and computer applications, vol. 98, pp. 27–42, 2017. 48, 49

[83] T. Hu, H. Chen, L. Huang, and X. Zhu, "A survey of mass data mining based on cloud-computing," in Anti-counterfeiting, Security, and Identification, pp. 1–4, IEEE, 2012. 52

[84] N. Golchha, "Big data–the information revolution," International journal of applied research, vol. 1, pp. 791–794, 2015. 52

[85] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, and I. Yaqoob, "Big iot data analytics: architecture, opportunities, and open research challenges," ieee access, vol. 5, pp. 5247–5261, 2017. 52, 56, 70, 162

[86] R. Tang and S. Fong, "Clustering big iot data by metaheuristic optimized mini-batch and parallel partition-based dgc in hadoop," Future Generation Computer Systems, vol. 86, pp. 1395–1412, 2018. 52, 66, 67

[87] V. Chaorasiya and A. SHRIVASTAVA, "A survey on big data: techniques and technologies," International Journal of Research and Development in Applied Science and Engineering, vol. 8, no. 1, pp. 1–4, 2015. 53

[88] E. G. Ularu, F. C. Puican, A. Apostu, M. Velicanu, et al., "Perspectives on big data and big data analytics," Database Systems Journal, vol. 3, no. 4, pp. 3–14, 2012. 53

[89] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U.

Khan, “The rise of “big data” on cloud computing: Review and open research issues,” Information systems, vol. 47, pp. 98–115, 2015. 53

[90] J. Gantz and D. Reinsel, “The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east,” IDC iView: IDC Analyze the future, vol. 2007, no. 2012, pp. 1–16, 2012. 53

[91] P. Tiainen et al., “New opportunities in electrical engineering as a result of the emergence of the internet of things,” Master’s thesis, 2016. 53

[92] S. Madden, “From databases to big data,” IEEE Internet Computing, vol. 16, no. 3, pp. 4–6, 2012. 53

[93] R. Omollo and S. Alago, “Data modeling techniques used for big data in enterprise networks,” 2020. 53

[94] J. Anuradha et al., “A brief introduction on big data 5vs characteristics and hadoop technology,” Procedia computer science, vol. 48, pp. 319–324, 2015. 54

[95] M. Lněnička, R. Máchová, J. Komárková, and I. Čermáková, “Components of big data analytics for strategic management of enterprise architecture,” in SMSIS 2017: Proceedings of the 12th International Conference on Strategic Management and its Support by Information Systems, Vysoká škola báňská-Technická univerzita Ostrava, 2017. 54

[96] A. Alexandru, C. Alexandru, D. Coardos, and E. Tudora, “Healthcare, big data and cloud computing,” management, vol. 1, no. 2, 2016. 54

[97] B. H. Malik, S. N. Cheema, I. Iqbal, Y. Mahmood, M. Ali, and A. Mudasser, “From cloud computing to fog computing (c2f): The key technology provides

services in health care big data," in MATEC Web of Conferences, vol. 189, p. 03010, EDP Sciences, 2018. 54

[98] P. Gulia and A. Chahal, "Big data analytics for iot," International Journal of Advanced Research in Engineering and Technology (IJARET), vol. 11, no. 6, 2020. 54

[99] N. Khan, M. Alsaqer, H. Shah, G. Badsha, A. A. Abbasi, and S. Salehian, "The 10 vs, issues and challenges of big data," in Proceedings of the 2018 international conference on big data and education, pp. 52–56, 2018. 55

[100] Z. Sun, K. Strang, and R. Li, "Big data with ten big characteristics," in Proceedings of the 2nd International Conference on Big Data Research, pp. 56–61, 2018. 55, 56

[101] M. Chen, S. Mao, Y. Zhang, V. C. Leung, et al., Big data: related technologies, challenges and future prospects, vol. 100. Springer, 2014. 56

[102] D. V. H. S. Kriti Srivastava, R. Shah, "Data mining using hierarchical agglomerative clustering algorithm in distributed cloud computing environment," International Journal of Computer Theory and Engineering, vol. 5, no. 3, 2013. 56

[103] T. Sajana, C. S. Rani, and K. Narayana, "A survey on clustering techniques for big data mining," Indian journal of Science and Technology, vol. 9, no. 3, pp. 1–12, 2016. 56, 58, 59, 61

[104] K. Krishna and M. N. Murty, "Genetic k-means algorithm," IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 29, no. 3, pp. 433–439, 1999. 57, 67, 119

[105] J. C. Bezdek, R. Ehrlich, and W. Full, “Fcm: The fuzzy c-means clustering algorithm,” Computers & geosciences, vol. 10, no. 2-3, pp. 191–203, 1984. 57, 67

[106] T. Zhang, R. Ramakrishnan, and M. Livny, “Birch: an efficient data clustering method for very large databases,” ACM sigmod record, vol. 25, no. 2, pp. 103–114, 1996. 58, 60, 67

[107] S. Guha, R. Rastogi, and K. Shim, “Cure: An efficient clustering algorithm for large databases,” ACM Sigmod record, vol. 27, no. 2, pp. 73–84, 1998. 58, 65, 67

[108] S. Guha, R. Rastogi, and K. Shim, “Rock: A robust clustering algorithm for categorical attributes,” Information systems, vol. 25, no. 5, pp. 345–366, 2000. 58, 59, 67

[109] Y. Rani[1] and H. Rohil, “A study of hierarchical clustering algorithm,” ter S & on Te SIT, vol. 2, p. 113, 2013. 58, 59, 67

[110] W. Wang, J. Yang, R. Muntz, et al., “Sting: A statistical information grid approach to spatial data mining,” in Vldb, vol. 97, pp. 186–195, Citeseer, 1997. 59, 60, 67

[111] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, “Automatic subspace clustering of high dimensional data for data mining applications,” in Proceedings of the 1998 ACM SIGMOD international conference on Management of data, pp. 94–105, 1998. 59, 60, 67

[112] S. Goil, H. Nagesh, and A. Choudhary, “Mafia: E±cient and scalable subspace clustering for very large data sets,” in Proc. 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Citeseer, pp. 443–452, Citeseer, 1999. 59, 60, 67

[113] T. Kohonen, "The self-organizing map," Proceedings of the IEEE, vol. 78, no. 9, pp. 1464–1480, 1990. 61, 67

[114] D. H. Fisher, "Knowledge acquisition via incremental conceptual clustering," Machine learning, vol. 2, no. 2, pp. 139–172, 1987. 61, 67

[115] B. Krose and P. v. d. Smagt, An introduction to neural networks. 2011. 61

[116] B. O. Reddy and M. Ussenaiah, "Literature survey on clustering techniques," IOSR Journal of Computer Engineering, vol. 3, no. 1, pp. 1–50, 2012. 61, 62, 66, 67, 69

[117] N. Sharma, A. Bajpai, and M. R. Litoriya, "Comparison the various clustering algorithms of weka tools," facilities, vol. 4, no. 7, pp. 78–80, 2012. 61

[118] G. S. Lee, "The effect of bias in data set for conceptual clustering algorithms," International journal of advanced smart convergence, vol. 8, no. 3, pp. 46–53, 2019. 62

[119] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., "A density-based algorithm for discovering clusters in large spatial databases with noise.," in kdd, vol. 96, pp. 226–231, 1996. 62, 67, 139, 193, 195

[120] m. . S. y. . . h. Farukh Hashmi, title = DBSCAN. 63

[121] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," ACM Sigmod record, vol. 28, no. 2, pp. 49–60, 1999. 64, 67

[122] D. WISHERT, "Mode analysis : a generalization of nearest neighbour which reduces chaining effects (with discussion)," Numerical Taxonomy, pp. 282–311, 1969. 64

[123] R. J. Campello, P. Kröger, J. Sander, and A. Zimek, “Density-based clustering,” Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 10, no. 2, p. e1343, 2020. 64

[124] R. Xu and D. Wunsch, “Survey of clustering algorithms,” IEEE Transactions on neural networks, vol. 16, no. 3, pp. 645–678, 2005. 65

[125] A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C.-T. Lin, “A review of clustering techniques and developments,” Neurocomputing, vol. 267, pp. 664–681, 2017. 65, 66, 67, 68, 69, 71

[126] J. H. Ward Jr, “Hierarchical grouping to optimize an objective function,” Journal of the American statistical association, vol. 58, no. 301, pp. 236–244, 1963. 65

[127] L. Rasyid and S. Andayani, “Review on clustering algorithms based on data type: towards the method for data combined of numeric-fuzzy linguistics,” in Journal of Physics: Conference Series, vol. 1097, p. 012082, IOP Publishing, 2018. 67

[128] G. Thilagavathi, D. Srivaishnavi, N. Aparna, et al., “A survey on efficient hierarchical algorithm used in clustering,” International Journal of Engineering, vol. 2, no. 9, pp. 165–176, 2013. 67, 68

[129] M. Cai and Y. Liang, “An improved cure algorithm,” in International conference on intelligence science, pp. 102–111, Springer, 2018. 67

[130] R. Subhashini and J. Akilandeswari, “A survey on ontology construction methodologies,” International Journal of Enterprise Computing and Business Systems, vol. 1, no. 1, pp. 60–72, 2011. 68

[131] J. Han, M. Kamber, and J. Pei, "Data mining: concepts and," Techniques (3rd ed), Morgan Kauffman, 2011. 68

[132] M. Carnein and H. Trautmann, "Optimizing data stream representation: An extensive survey on stream clustering algorithms," Business & Information Systems Engineering, vol. 61, no. 3, pp. 277–297, 2019. 69

[133] A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C. A. C. Coello, "A survey of multiobjective evolutionary algorithms for data mining: Part i," IEEE Transactions on Evolutionary Computation, vol. 18, no. 1, pp. 4–19, 2013. 69

[134] A. Gosain and M. Bhugra, "A comprehensive survey of association rules on quantitative data in data mining," in 2013 IEEE Conference on Information & Communication Technologies, pp. 1003–1008, IEEE, 2013. 70

[135] C. Luo and S. M. Chung, "Efficient mining of maximal sequential patterns using multiple samples," in Proceedings of the 2005 SIAM International Conference on Data Mining, pp. 415–426, SIAM, 2005. 71

[136] Z. Yang and M. Kitsuregawa, "Lapin-spam: An improved algorithm for mining sequential pattern," in 21st International Conference on Data Engineering Workshops (ICDEW'05), pp. 1222–1222, IEEE, 2005. 71

[137] V. Estivill-Castro, "Why so many clustering algorithms: a position paper," ACM SIGKDD explorations newsletter, vol. 4, no. 1, pp. 65–75, 2002. 71

[138] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in Proceedings of the thirtieth annual ACM symposium on Theory of computing, pp. 604–613, 1998. 75

[139] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in Proceedings of the twentieth annual symposium on Computational geometry, pp. 253–262, 2004. 75

[140] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: efficient indexing for high-dimensional similarity search," in 33rd International Conference on Very Large Data Bases, VLDB 2007, pp. 950–961, Association for Computing Machinery, Inc, 2007. 75

[141] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Efficient and accurate nearest neighbor and closest pair search in high-dimensional space," ACM Transactions on Database Systems (TODS), vol. 35, no. 3, pp. 1–46, 2010. 75

[142] B. Zheng, Z. Xi, L. Weng, N. Q. V. Hung, H. Liu, and C. S. Jensen, "Pm-lsh: A fast and accurate lsh framework for high-dimensional approximate nn search," Proceedings of the VLDB Endowment, vol. 13, no. 5, pp. 643–655, 2020. 75, 76, 77, 121

[143] J. Gan, J. Feng, Q. Fang, and W. Ng, "Locality-sensitive hashing scheme based on dynamic collision counting," in Proceedings of the 2012 ACM SIGMOD international conference on management of data, pp. 541–552, 2012. 75, 121

[144] Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng, "Query-aware locality-sensitive hashing for approximate nearest neighbor search," Proceedings of the VLDB Endowment, vol. 9, no. 1, pp. 1–12, 2015. 75, 121

[145] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil, "The log-structured merge-tree (lsm-tree)," Acta Informatica, vol. 33, no. 4, pp. 351–385, 1996. 75

[146] G. Irie, Z. Li, X.-M. Wu, and S.-F. Chang, "Locally linear hashing for extracting non-linear manifolds," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2115–2122, 2014. 76

[147] N. Kraus, D. Carmel, I. Keidar, and M. Orenbach, "Nearbucket-lsh: Efficient similarity search in p2p networks," in International conference on similarity search and applications, pp. 236–249, Springer, 2016. 76, 121

[148] J. Liao, D. Yang, T. Li, Q. Qi, J. Wang, and H. Sun, "Fusion feature for lsh-based image retrieval in a cloud datacenter," Multimedia Tools and Applications, vol. 75, no. 23, pp. 15405–15427, 2016. 76, 121

[149] Y.-T. Chuang, C.-Y. Yu, and Q.-W. Wu, "Dslm: a decentralized search for large and mobile networks," The Journal of Supercomputing, vol. 74, no. 2, pp. 738–767, 2018. 77, 121

[150] J. Wu, L. Shen, and L. Liu, "Lsh-based distributed similarity indexing with load balancing in high-dimensional space," The Journal of Supercomputing, vol. 76, no. 1, pp. 636–665, 2020. 77, 121

[151] Y. Yang, F. Shen, H. T. Shen, H. Li, and X. Li, "Robust discrete spectral hashing for large-scale image semantic indexing," IEEE Transactions on Big Data, vol. 1, no. 4, pp. 162–171, 2015. 77, 78, 122

[152] Z. Kouahla, A.-E. Benrazek, M. A. Ferrag, B. Farou, H. Seridi, M. Kurulay, A. Anjum, and A. Asheralieva, "A survey on big iot data indexing: Potential solutions, recent advancements, and open issues," Future Internet, vol. 14, no. 1, p. 19, 2021. 77, 78, 108

[153] F. S. Patel and D. Kasat, "Hashing based indexing techniques for content based image retrieval: A survey," in 2017 International Conference on

Innovative Mechanisms for Industry Applications (ICIMIA), pp. 279–283, IEEE, 2017. 77

[154] D. Cai, "A revisit of hashing algorithms for approximate nearest neighbor search," IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 6, pp. 2337–2348, 2019. 77

[155] L. Xie, J. Shen, J. Han, L. Zhu, and L. Shao, "Dynamic multi-view hashing for online image retrieval," IJCAI, 2017. 78, 122

[156] A. Mourão and J. Magalhães, "Towards cloud distributed image indexing by sparse hashing," in Proceedings of the 2019 on International Conference on Multimedia Retrieval, pp. 288–296, 2019. 78, 122

[157] D. Zhai, X. Liu, X. Ji, D. Zhao, S. Satoh, and W. Gao, "Supervised distributed hashing for large-scale multimedia retrieval," IEEE Transactions on Multimedia, vol. 20, no. 3, pp. 675–686, 2017. 78, 122

[158] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," IEEE transactions on pattern analysis and machine intelligence, vol. 35, no. 12, pp. 2916–2929, 2012. 79, 122

[159] K. He, F. Wen, and J. Sun, "K-means hashing: An affinity-preserving quantization method for learning binary compact codes," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2938–2945, 2013. 79, 122

[160] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," arXiv preprint arXiv:1408.2927, 2014. 79, 120, 122

[161] J. Li, J. Cheng, F. Yang, Y. Huang, Y. Zhao, X. Yan, and R. Zhao, “Losha: A general framework for scalable locality sensitive hashing,” in Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 635–644, 2017. 79, 122

[162] J. L. Bentley, “Multidimensional binary search trees in database applications,” IEEE Transactions on Software Engineering, no. 4, pp. 333–340, 1979. 79, 80, 81, 83, 124

[163] H.-K. Ahn, N. Mamoulis, and H. M. Wong, “A survey on multidimensional access methods,” 2001. 80

[164] M. d. Berg, M. v. Kreveld, M. Overmars, and O. Schwarzkopf, “Computational geometry,” in Computational geometry, pp. 1–17, Springer, 1997. 80

[165] J. B. Rosenberg, “Geographical data structures compared: A study of data structures supporting region queries,” IEEE transactions on computer-aided design of integrated circuits and systems, vol. 4, no. 1, pp. 53–67, 1985. 80

[166] J. T. Robinson, “The kdb-tree: a search structure for large multidimensional dynamic indexes,” in Proceedings of the 1981 ACM SIGMOD international conference on Management of data, pp. 10–18, 1981. 80, 124

[167] D. Comer, “Ubiquitous b-tree,” ACM Computing Surveys (CSUR), vol. 11, no. 2, pp. 121–137, 1979. 80, 83, 124

[168] R. A. Finkel and J. L. Bentley, “Quad trees a data structure for retrieval on composite keys,” Acta informatica, vol. 4, no. 1, pp. 1–9, 1974. 81, 124

[169] A. A. Visheratin, K. D. Mukhina, A. K. Visheratina, D. Nasonov, and A. V. Boukhanovsky, “Multiscale event detection using convolutional quadtrees

and adaptive geogrids," in Proceedings of the 2nd ACM SIGSPATIAL Workshop on Analytics for Local Events and News, pp. 1–10, 2018. 81

[170] A. Watve, S. Pramanik, S. Shahid, C. R. Meiners, and A. X. Liu, "Topological transformation approaches to database query processing," IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 5, pp. 1438–1451, 2014. 81

[171] M. R. Abbasifard, B. Ghahremani, and H. Naderi, "A survey on nearest neighbor search methods," International Journal of Computer Applications, vol. 95, no. 25, 2014. 82

[172] N. Katayama and S. Satoh, "The sr-tree: An index structure for high-dimensional nearest neighbor queries," ACM Sigmod Record, vol. 26, no. 2, pp. 369–380, 1997. 82, 124

[173] S. Günnemann, H. Kremer, D. Lenhard, and T. Seidl, "Subspace clustering for indexing high dimensional data: a main memory index based on local reductions and individual multi-representations," in Proceedings of the 14th International Conference on Extending Database Technology, pp. 237–248, 2011. 82, 124

[174] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The r+-tree: A dynamic index for multi-dimensional objects.," tech. rep., 1987. 82, 124

[175] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: An efficient and robust access method for points and rectangles," in Proceedings of the 1990 ACM SIGMOD international conference on Management of data, pp. 322–331, 1990. 82, 124

[176] S. Sprenger, P. Schäfer, and U. Leser, "Bb-tree: A practical and efficient

main-memory index structure for multidimensional workloads.," in EDBT, pp. 169–180, 2019. 82, 83, 124, 132, 144, 207, 208, 213, 222

[177] S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The x-tree: An index structure for high-dimensional data," in Very Large Data-Bases, pp. 28–39, 1996. 82, 83, 104, 124

[178] S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, "Indexing the positions of continuously moving objects," in Proceedings of the 2000 ACM SIGMOD international conference on Management of data, pp. 331–342, 2000. 83, 124

[179] Y. Tao, D. Papadias, and J. Sun, "The tpr*-tree: An optimized spatio-temporal access method for predictive queries," in Proceedings 2003 VLDB conference, pp. 790–801, Elsevier, 2003. 83, 124

[180] Z. He, C. Wu, G. Liu, Z. Zheng, and Y. Tian, "Decomposition tree: A spatio-temporal indexing method for movement big data," Cluster Computing, vol. 18, no. 4, pp. 1481–1492, 2015. 83, 124

[181] R. Bayer, "The universal b-tree for multidimensional indexing: General concepts," in International Conference on Worldwide Computing and Its Applications, pp. 198–209, Springer, 1997. 83, 84, 124

[182] V. Srinivasan and M. J. Carey, "Performance of b+ tree concurrency control algorithms," The VLDB Journal, vol. 2, no. 4, pp. 361–406, 1993. 84, 87, 124

[183] H.-Y. Lin, "Using compressed index structures for processing moving objects in large spatio-temporal databases," Journal of Systems and Software, vol. 85, no. 1, pp. 167–177, 2012. 84, 124

[184] S. Wu and K.-L. Wu, “An indexing framework for efficient retrieval on the cloud.,” IEEE Data Eng. Bull., vol. 32, no. 1, pp. 75–82, 2009. 85, 125

[185] X. Zhang, J. Ai, Z. Wang, J. Lu, and X. Meng, “An efficient multi-dimensional index for cloud data management,” in Proceedings of the first international workshop on Cloud data management, pp. 17–24, 2009. 85, 125, 155

[186] A. Papadopoulos and D. Katsaros, “A-tree: Distributed indexing of multidimensional data for cloud computing environments,” in 2011 IEEE Third International Conference on Cloud Computing Technology and Science, pp. 407–414, IEEE, 2011. 85, 125

[187] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422–426, 1970. 85

[188] J. H. Friedman, J. L. Bentley, and R. A. Finkel, “An algorithm for finding best matches in logarithmic expected time,” ACM Transactions on Mathematical Software (TOMS), vol. 3, no. 3, pp. 209–226, 1977. 85

[189] S. Wu, D. Jiang, B. C. Ooi, and K.-L. Wu, “Efficient b-tree based indexing for cloud data processing,” Proceedings of the VLDB Endowment, vol. 3, no. 1-2, pp. 1207–1218, 2010. 85, 86, 125

[190] H. V. Jagadish, B. C. Ooi, M. C. Rinard, and Q. H. Vu, “Baton: A balanced tree structure for peer-to-peer networks,” 2006. 85

[191] J. Wang, S. Wu, H. Gao, J. Li, and B. C. Ooi, “Indexing multi-dimensional data in a cloud system,” in Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 591–602, 2010. 85, 86, 125

[192] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content-addressable network,” in Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 161–172, 2001. 85

[193] Y. Hong, Q. Tang, X. Gao, B. Yao, G. Chen, and S. Tang, “Efficient r-tree based indexing scheme for server-centric cloud storage system,” IEEE Transactions on Knowledge and Data Engineering, vol. 28, no. 6, pp. 1503–1517, 2016. 86, 125, 130

[194] C. Feng, X. Yang, F. Liang, X.-H. Sun, and Z. Xu, “Lcindex: a local and clustering index on distributed ordered tables for flexible multi-dimensional range queries,” in 2015 44th International Conference on Parallel Processing, pp. 719–728, IEEE, 2015. 86, 125, 130

[195] Y. Gao, X. Gao, Y. Zhu, and G. Chen, “An efficient and scalable multi-dimensional indexing scheme for modular data centers,” Data & Knowledge Engineering, vol. 123, p. 101729, 2019. 87, 125, 130

[196] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, “Bcube: a high performance, server-centric network architecture for modular data centers,” in Proceedings of the ACM SIGCOMM 2009 conference on Data communication, pp. 63–74, 2009. 87

[197] X. Gao, Y. Gao, Y. Zhu, and G. Chen, “U 2-tree: A universal two-layer distributed indexing scheme for cloud storage system,” IEEE/ACM Transactions on Networking, vol. 27, no. 1, pp. 201–213, 2019. 87, 125, 130

[198] S. Wang, D. Maier, and B. C. Ooi, “Lightweight indexing of observational data in log-structured storage,” Proceedings of the VLDB Endowment, vol. 7, no. 7, pp. 529–540, 2014. 87, 88, 125

[199] Y. Fathy, P. Barnaghi, and R. Tafazolli, "Large-scale indexing, discovery, and ranking for the internet of things (iot)," ACM Computing Surveys (CSUR), vol. 51, no. 2, pp. 1–53, 2018. 88, 125, 162

[200] Y. Zou, J. Liu, S. Wang, L. Zha, and Z. Xu, "Ccindex: A complemental clustering index on distributed ordered tables for multi-dimensional range queries," in IFIP International Conference on Network and Parallel Computing, pp. 247–261, Springer, 2010. 88, 125

[201] Y. Ma, J. Rao, W. Hu, X. Meng, X. Han, Y. Zhang, Y. Chai, and C. Liu, "An efficient index for massive iot data in cloud environment," in Proceedings of the 21st ACM international conference on Information and knowledge management, pp. 2129–2133, 2012. 88, 125, 132, 144, 207, 222

[202] C. Y. Huang and Y. J. Chang, "An adaptively multi-attribute index framework for big iot data," Computers & Geosciences, p. 104841, 2021. 89, 125

[203] Z. Ding, J. Xu, and Q. Yang, "Seaclouddm: a database cluster framework for managing and querying massive heterogeneous sensor sampling data," The Journal of Supercomputing, vol. 66, no. 3, pp. 1260–1284, 2013. 89, 125

[204] X. Chen, J. Xu, R. Zhou, P. Zhao, C. Liu, J. Fang, and L. Zhao, "S2r-tree: a pivot-based indexing structure for semantic-aware spatial keyword search," GeoInformatica, vol. 24, no. 1, pp. 3–25, 2020. 90, 123, 125

[205] J. Xia, S. Huang, S. Zhang, X. Li, J. Lyu, W. Xiu, and W. Tu, "Dapr-tree: a distributed spatial data indexing scheme with data access patterns to support digital earth initiatives," International Journal of Digital Earth, vol. 13, no. 12, pp. 1656–1671, 2020. 90, 123, 125

[206] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer, "An asymptotically optimal multiversion b-tree," The VLDB Journal, vol. 5, no. 4, pp. 264–275, 1996. 91

[207] D. Kumar et al., "Dpiscan: Distributed and parallel architecture with indexing for structural clustering of massive dynamic graphs," International Journal of Data Science and Analytics, pp. 1–25, 2022. 91, 125

[208] Y. Mao, E. Kohler, and R. T. Morris, "Cache craftiness for fast multicore key-value storage," in Proceedings of the 7th ACM european conference on Computer Systems, pp. 183–196, 2012. 91

[209] S. V. Limkar and R. K. Jha, "A novel method for parallel indexing of real time geospatial big data generated by iot devices," Future generation computer systems, vol. 97, pp. 433–452, 2019. 92, 125, 219

[210] F. Hu, C. Yang, Y. Jiang, Y. Li, W. Song, D. Q. Duffy, J. L. Schnase, and T. Lee, "A hierarchical indexing strategy for optimizing apache spark with hdfs to efficiently query big geospatial raster data," International Journal of Digital Earth, vol. 13, no. 3, pp. 410–428, 2020. 92, 155

[211] Q.-T. Doan, A. Kayes, W. Rahayu, and K. Nguyen, "Integration of iot streaming data with efficient indexing and storage optimization," IEEE Access, vol. 8, pp. 47456–47467, 2020. 92, 93, 125

[212] S. S. Bavirthi et al., "An approach for combining spatial and textual skyline querying using indexing mechanism," Turkish Journal of Computer and Mathematics Education (TURCOMAT), vol. 12, no. 11, pp. 672–680, 2021. 93, 125

[213] S. S. Bavirthi and K. Supreethi, "An efficient framework for spatio-textual

skyline querying and minimizing search space using r+ tree indexing technique," International Journal of Information Technology, pp. 1–9, 2022. 93, 125

[214] D. Abadi, A. Ailamaki, D. Andersen, P. Bailis, M. Balazinska, P. Bernstein, P. Boncz, S. Chaudhuri, A. Cheung, A. Doan, et al., "The seattle report on database research," ACM SIGMOD Record, vol. 48, no. 4, pp. 44–53, 2020. 93

[215] G. Weintraub, E. Gudes, and S. Dolev, "Needle in a haystack queries in cloud data lakes.," in EDBT/ICDT Workshops, 2021. 93, 125

[216] S. Wan, Y. Zhao, T. Wang, Z. Gu, Q. H. Abbasi, and K.-K. R. Choo, "Multi-dimensional data indexing and range query processing via voronoi diagram for internet of things," Future Generation Computer Systems, vol. 91, pp. 382–391, 2019. 94, 125

[217] I. Kalantari and G. McDonald, "A data structure and an algorithm for the nearest point problem," IEEE Transactions on Software Engineering, no. 5, pp. 631–634, 1983. 95, 127

[218] H. Noltemeier, K. Verbarg, and C. Zirkelbach, "Monotonous bisector* trees—a tool for efficient partitioning of complex scenes of geometric objects," Data structures and efficient algorithms, pp. 186–203, 1992. 96, 127

[219] F. Dehne and H. Noltemeier, "Voronoi trees and clustering problems," Information Systems, vol. 12, no. 2, pp. 171–175, 1987. 96, 127

[220] J. K. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," Information processing letters, vol. 40, no. 4, pp. 175–179, 1991. 97, 115, 127, 193, 195

[221] Z. Kouahla and J. Martinez, "A new intersection tree for content-based image retrieval," in 2012 10th International Workshop on Content-Based Multimedia Indexing (CBMI), pp. 1–6, IEEE, 2012. 97, 103, 104, 127

[222] S. Brin, "Near neighbor search in large metric spaces," 1995. 98, 127

[223] G. Navarro and R. Uribe-Paredes, "Fully dynamic metric access methods based on hyperplane partitioning," Information Systems, vol. 36, no. 4, pp. 734–747, 2011. 98, 127

[224] M. Antol and V. Dohnal, "Bm-index: balanced metric space index based on weighted voronoi partitioning," in European Conference on Advances in Databases and Information Systems, pp. 337–353, Springer, 2019. 99

[225] A. Moriyama, L. S. Rodrigues, L. C. Scabora, M. T. Cazzolato, A. J. Traina, and C. Traina Jr, "Vd-tree: how to build an efficient and fit metric access method using voronoi diagrams," in Proceedings of the 36th Annual ACM Symposium on Applied Computing, pp. 327–335, 2021. 99, 127

[226] C. Traina, A. Traina, B. Seeger, and C. Faloutsos, "Slim-trees: High performance metric trees minimizing overlap between nodes," in International Conference on Extending Database Technology, pp. 51–65, Springer, 2000. 99, 110, 127

[227] R. Mao, S. Liu, H. Xu, D. Zhang, and D. P. Miranker, "On data partitioning in tree structure metric-space indexes," in International Conference on Database Systems for Advanced Applications, pp. 141–155, Springer, 2014. 100, 109

[228] T. Bozkaya and M. Ozsoyoglu, "Distance-based indexing for high-dimensional metric spaces," in Proceedings of the 1997 ACM SIGMOD

international conference on Management of data, pp. 357–368, 1997. 100, 101

[229] P. N. Yianilos, "Data structures and algorithms for nearest neighbor," in Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms, vol. 66, p. 311, SIAM, 1993. 100, 116, 127

[230] X. Zhou, G. Wang, J. X. Yu, and G. Yu, "M+-tree: A new dynamical multi-dimensional index for metric spaces," in Proceedings of the 14th Australasian database conference-Volume 17, pp. 161–168, 2003. 100

[231] T. Bozkaya and M. Ozsoyoglu, "Indexing large metric spaces for similarity search queries," ACM Transactions on Database Systems (TODS), vol. 24, no. 3, pp. 361–404, 1999. 100, 127

[232] I. R. V. Pola, C. Traina, and A. J. M. Traina, "The mm-tree: A memory-based metric tree without overlap between nodes," in East European Conference on Advances in Databases and Information Systems, pp. 157–171, Springer, 2007. 101, 102, 127

[233] C. C. M. Carélo, I. R. V. Pola, R. R. Ciferri, A. J. M. Traina, C. Traina Jr, and C. D. de Aguiar Ciferri, "Slicing the metric space to provide quick indexing of complex data in the main memory," Information Systems, vol. 36, no. 1, pp. 79–98, 2011. 101, 102, 103, 127

[234] Z. Kouahla, A. Anjum, S. Akram, T. Saba, and J. Martinez, "Xm-tree: data driven computational model by using metric extended nodes with non-overlapping in high-dimensional metric spaces," Computational and Mathematical Organization Theory, vol. 25, no. 2, pp. 196–223, 2019. 104, 105, 127

[235] I. R. V. Pola, C. Traina Jr, and A. J. M. Traina, "The nobh-tree: Improving in-memory metric access methods by using metric hyperplanes with non-overlapping nodes," Data & Knowledge Engineering, vol. 94, pp. 65–88, 2014. 105, 106, 127

[236] V. Dohnal, C. Gennaro, P. Savino, and P. Zezula, "D-index: Distance searching index for metric data sets," Multimedia Tools and Applications, vol. 21, no. 1, pp. 9–33, 2003. 106, 109, 127

[237] L. Chen, Y. Gao, X. Song, Z. Li, Y. Zhu, X. Miao, and C. S. Jensen, "Indexing metric spaces for exact similarity search," ACM Computing Surveys (CSUR), 2020. 106, 107, 219

[238] V. Dohnal, C. Gennaro, and P. Zezula, "Similarity join in metric spaces using ed-index," in International Conference on Database and Expert Systems Applications, pp. 484–493, Springer, 2003. 106, 109, 127

[239] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, "idistance: An adaptive b+-tree based indexing method for nearest neighbor search," ACM Transactions on Database Systems (TODS), vol. 30, no. 2, pp. 364–397, 2005. 106, 108, 109, 116, 127

[240] D. Novak, M. Batko, and P. Zezula, "Metric index: An efficient and scalable solution for precise and approximate similarity search," Information Systems, vol. 36, no. 4, pp. 721–733, 2011. 107, 108, 109, 127

[241] L. Chen, Y. Gao, X. Li, C. S. Jensen, and G. Chen, "Efficient metric indexing for similarity search and similarity joins," IEEE Transactions on Knowledge and Data Engineering, vol. 29, no. 3, pp. 556–571, 2015. 108, 109, 127

[242] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method

for similarity search in metric spaces," in Vldb, vol. 97, pp. 426–435, 1997. 109, 110, 113

[243] M. Batko, C. Gennaro, and P. Zezula, "Similarity grid for searching in metric spaces," in Peer-To-peer, grid, and service-orientation in digital library architectures, pp. 25–44, Springer, 2005. 109, 116, 127

[244] T. Skopal, J. Pokornỳ, M. Krátkỳ, and V. Snášel, "Revisiting m-tree building principles," in East European Conference on Advances in Databases and Information Systems, pp. 148–162, Springer, 2003. 110

[245] M. R. Vieira, C. Traina, F. J. Chino, and A. J. Traina, "Dbm-tree: A dynamic metric access method sensitive to local density data," in In SBBD, Citeseer, 2004. 111, 127

[246] A. Ocsa and E. Cuadros-Vargas, "Dbm*-tree: an efficient metric access method," in Proceedings of the 45th annual southeast regional conference, pp. 401–406, 2007. 111, 112, 127

[247] S. Jin, O. Kim, and W. Feng, "M x-tree: A double hierarchical metric index with overlap reduction," in International Conference on Computational Science and Its Applications, pp. 574–589, Springer, 2013. 112, 127, 132, 144, 207, 208, 213, 222

[248] H. Razente and M. C. Nardini Barioni, "Storing data once in m-tree and pm-tree," in International Conference on Similarity Search and Applications, pp. 18–31, Springer, 2019. 113, 127

[249] J. P. Bachmann, "The superm-tree: Indexing metric spaces with sized objects," arXiv preprint arXiv:1901.11453, 2019. 113, 127

[250] S. Brinis, C. Traina, and A. J. Traina, “Hollow-tree: a metric access method for data with missing values,” Journal of Intelligent Information Systems, vol. 53, no. 3, pp. 481–508, 2019. 113, 127

[251] E. Vidal, “New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (aesa),” Pattern Recognition Letters, vol. 15, no. 1, pp. 1–7, 1994. 114, 127

[252] E. V. Ruiz, “An algorithm for finding nearest neighbours in (approximately) constant average time,” Pattern Recognition Letters, vol. 4, no. 3, pp. 145–157, 1986. 114, 115

[253] M. L. Micó, J. Oncina, and E. Vidal, “A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements,” Pattern Recognition Letters, vol. 15, no. 1, pp. 9–17, 1994. 114, 127

[254] Y. Hanyf and H. Silkan, “A queries-based structure for similarity searching in static and dynamic metric spaces,” Journal of King Saud University-Computer and Information Sciences, vol. 32, no. 2, pp. 188–196, 2020. 114

[255] G. Ruiz, F. Santoyo, E. Chávez, K. Figueroa, and E. S. Tellez, “Extreme pivots for faster metric indexes,” in International Conference on Similarity Search and Applications, pp. 115–126, Springer, 2013. 114

[256] Y. Hanyf, H. Silkan, and H. Labani, “An improvable structure for similarity searching in metric spaces: application on image databases,” in 2016 13th International Conference on Computer Graphics, Imaging and Visualization (CGiV), pp. 67–72, IEEE, 2016. 114, 127

[257] L. Micó, J. Oncina, and R. C. Carrasco, “A fast branch & bound nearest neighbour classifier in metric spaces,” Pattern Recognition Letters, vol. 17, no. 7, pp. 731–739, 1996. 115

[258] M. Batko, D. Novak, F. Falchi, and P. Zezula, “Scalability comparison of peer-to-peer similarity search structures,” Future Generation Computer Systems, vol. 24, no. 8, pp. 834–848, 2008. 115, 116, 128

[259] V. Dohnal, J. Sedmidubsky, P. Zezula, and D. Novák, “Similarity searching: Towards bulk-loading peer-to-peer networks,” in 2008 IEEE 24th International Conference on Data Engineering Workshop, pp. 378–385, IEEE, 2008. 116, 128

[260] H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” in ACM SIGCOMM, Citeseer, 2001. 116

[261] F. Falchi, C. Gennaro, and P. Zezula, “A content–addressable network for similarity search in metric spaces,” in Databases, Information Systems, and Peer-to-Peer Computing, pp. 98–110, Springer, 2006. 116, 128

[262] Z. Kouahla and A. Anjum, “A parallel implementation of ghb tree,” in IFIP International Conference on Computational Intelligence and Its Applications, pp. 47–55, Springer, 2018. 117, 128, 137, 193, 195

[263] K. Yang, X. Ding, Y. Zhang, L. Chen, B. Zheng, and Y. Gao, “Distributed similarity queries in metric spaces,” Data Science and Engineering, vol. 4, no. 2, pp. 93–108, 2019. 118, 128, 219

[264] P. Do and T. H. Phan, “A distributed m-tree for similarity search in large multimedia database on spark,” in Handbook of Research on Multimedia Cyber Security, pp. 146–164, IGI Global, 2020. 119, 128

[265] O. Jafari, P. Nagarkar, and J. Montaño, "Improving locality sensitive hashing by efficiently finding projected nearest neighbors," in International Conference on Similarity Search and Applications, pp. 323–337, Springer, 2020. 121

[266] Z. Yao, J. Zhang, and J. Feng, "Nv-qalsh: An nvm-optimized implementation of query-aware locality-sensitive hashing," in International Conference on Database and Expert Systems Applications, pp. 58–69, Springer, 2021. 121

[267] X. Lu, L. Zhu, Z. Cheng, J. Li, X. Nie, and H. Zhang, "Flexible online multi-modal hashing for large-scale multimedia retrieval," in Proceedings of the 27th ACM international conference on multimedia, pp. 1129–1137, 2019. 122

[268] M. Etemadi, M. Ghobaei-Arani, and A. Shahidinejad, "Resource provisioning for iot services in the fog computing environment: An autonomic approach," Computer Communications, vol. 161, pp. 109–131, 2020. 132

[269] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," ACM Transactions on Database Systems (TODS), vol. 42, no. 3, pp. 1–21, 2017. 139

[270] "Geographical coordinates." http://data.public.lu/fr/datasets/r/a7d551d7-f374-491aab93-63715b98e6dd,2019. 142, 212

[271] "Gps trajectories data set." https://archive.ics.uci.edu/ml/datasets/GPS+Trajectories. 142, 173

[272] "Ward." https://people.eecs.berkeley.edu/~yang/software/WAR/WARD1.zip,2019. 142, 173

[273] A. Y. Yang, R. Jafari, S. S. Sastry, and R. Bajcsy, “Distributed recognition of human actions using wearable motion sensor networks,” Journal of Ambient Intelligence and Smart Environments, vol. 1, no. 2, pp. 103–115, 2009. 142, 173

[274] “Smart home data.” `https://www.kaggle.com/cnrieiit/mqttset/version/1`. 142

[275] I. Vaccari, G. Chiola, M. Aiello, M. Mongelli, and E. Cambiaso, “Mqttset, a new dataset for machine learning techniques on mqtt,” Sensors, vol. 20, no. 22, p. 6578, 2020. 143, 197

[276] S. O. Al-mamory and Z. M. Algelal, “A modified dbscan clustering algorithm for proactive detection of ddos attacks,” in 2017 Annual Conference on New Trends in Information & Communications Technology Applications (NTICT), pp. 304–309, IEEE, 2017. 165

[277] N. Mehta and S. Dang, “A review of clustering techiques in various applications for effective data mining,” International Journal of Research in IT & Management, ISSN, pp. 2231–4334, 2011. 166

[278] K. Khettabi, Z. Kouahla, B. Farou, H. Seridi, and M. A. Ferrag, “Clustering and parallel indexing of big iot data in the fog-cloud computing level,” Transactions on Emerging Telecommunications Technologies, p. e4484. 166, 170

[279] T. Liu, S. Qu, and K. Zhang, “A clustering algorithm for automatically determining the number of clusters based on coefficient of variation,” in Proceedings of the 2nd International Conference on Big Data Research, pp. 100–106, 2018. 166

[280] K. Khettabi, Z. Kouahla, B. Farou, H. Seridi, and M. A. Ferrag, "Clustering and parallel indexing of big iot data in the fog-cloud computing level," Transactions on Emerging Telecommunications Technologies, p. e4484, 2022. 168

[281] R. Rossi and N. Ahmed, "The network data repository with interactive graph analytics and visualization," in Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015. 173

[282] H.-Y. Wu and C.-R. Lee, "Energy efficient scheduling for heterogeneous fog computing architectures," in 2018 IEEE 42nd annual computer software and applications conference (COMPSAC), vol. 1, pp. 555–560, IEEE, 2018. 177

[283] K. Zhang, W. Zhou, S. Sun, and B. Li, "Multiple complementary inverted indexing based on multiple metrics," Multimedia Tools and Applications, vol. 78, no. 6, pp. 7727–7747, 2019. 190

[284] `https://zenodo.org/record/4972594#.YjBi3tXMLIX`.

[285] `https://http://ieee-dataport.org/documents/edge-iiotset-new\`
`comprehensive-realistic-cyber-security-dataset-iot-\`
`and-iiot-applications`.

[286] `https://www.kaggle.com/ranakrc/smart-building-system`.

[287] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in AAAI, 2015.

[288] "Road." `https://networkrepository.com/road.php`.

List of Publications

International Publications

1. K. Khettabi, Z. Kouahla, B. Farou, H. Seridi and M.A. Ferrag, **Clustering and parallel indexing of big IoT data in the fog-cloud computing level**, *Transactions on Emerging Telecommunications Technologies*, p.e4484,2022, https://doi.org/10.1002/ett.4484.
2. K. Khettabi, Z. Kouahla, B. Farou, H. Seridi and M.A. Ferrag, **A new method for indexing continuous IoT data flows in metric space**, *Internet Technology Letters*, p.e391,2022 https://doi.org/10.1002/itl2.391.
3. K. Khettabi, Z. Kouahla, B. Farou, H. Seridi and M.A. Ferrag, **Efficient Method for Continuous IoT Data Stream Indexing in the Fog-Cloud Computing Level**, *IEEE Transactions on Services Computing.* Current statue: Minor Revision (Under Review).

International Communications

K. Khettabi, Z. Kouahla, B. Farou and H. Seridi, **QCCF-tree: A New Efficient IoT Big Data Indexing Method at the Fog-Cloud Computing Level**, *2021 IEEE International Smart Cities Conference (ISC2)*, p. 1-7, 2021, https://doi: 10.1109/ISC253183.2021.9562836.

National Communications

K. Khettabi, Z. Kouahla, B. Farou and H. Seridi, **Comparison of indexing methods for large IoT data sets**. *Informatics and Applied Mathematics (IAM2020)* Guelma, Algeria, 2020.