GeoSharding: Optimization of data partitioning in sharded georeferenced databases

João Pedro Martins Graça

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Orientadores: Professor João Nuno Oliveira Silva
Professora Laura Ricci

Júri

Orientador: Professor João Nuno Oliveira Silva

Outubro de 2016

Acknowledgments

I would first like to thank my thesis adviser Professor João Nuno Silva, since he was always available to help me whenever I needed. Besides all the attention throughout the thesis work, I must deeply thank him for helping me to study abroad and to start this thesis' investigation in a foreign university.

In addition, I would like to express my gratitude to Professor Laura Ricci of the Computer Science Department at University of Pisa for all the guidance she gave me abroad and all the help regarding the investigation work of this master thesis.

Finally, I would like to thank my parents, grandparents and brother for providing me with continuous encouragement and support throughout my years as student. Certainly, this accomplishment would not have been possible without them.

Abstract

Currently the volume of geo-referenced data in the web, data associated with a physical location, is continuously growing (like data on Social Networks, for example).

NoSQL databases use sharding and partitioning algorithms to load balance and optimize data accesses. These mechanisms rely on the database structure to define the boundaries of the sets stored by each node, and assume some sort of access locality to data. However, some of these partitioning and spatial indexing mechanisms are not very efficient and could be optimized.

The partitioning policy developed and proposed in this master thesis exploits the geographical positions of the data to define a new sharding mechanism, called GeoSharding, that uses Voronoi diagrams as the basic structure of spatial indexing.

The initial goals of the work described in this document were to study the state of the art of the NoSQL databases with respect to data partition algorithms, the state of the art of the geographic data storage mechanisms and the georeferenced data partition mechanisms' possibilities. The main objective of the realized work was to define an algorithm to simulate GeoSharding and show how the the proposed policy can optimize data partitioning in a NoSQL system.

The results obtained show that this method can optimize data partitioning regarding spatial queries performance, load balancing across the network shards and data replication in systems that store and manage spatial data.

Keywords

Sharding; Georeferenced data; NoSQL; Big data;

Resumo

Actualmente o volume de dados georeferenciados, dados associados a uma posição física, está em contínuo crescimento (como por exemplo, fotografias em redes sociais).

As bases de dados NoSQL utilizam o sharding e algoritmos de particionamento para distribuir equilibradamente a informação dos dados e optimizar o acesso aos mesmos. Estes mecanismos dependem da estrutura do sistema de armazenamento para definir os limites dos grupos de dados guardados em cada shard, definindo assim uma espécie de localidade de acesso aos dados. No entanto, alguns destes mecanismos de particionamento e de indexação espacial não são muito eficientes e podem ser optimizados.

A estretégia de particionamento desenvolvida e proposta nesta dissertação explora a posição geográfica dos dados para definir um novo mecanismo de sharding, chamado GeoSharding, que utiliza diagramas de Voronoi como estrutura base do mecanismo de indexação espacial.

Os objectivos iniciais desta tese de mestrado foram estudar o estado de arte dos sistemas de armazenamento NoSQL e dos seus algoritmos de particionamento, o estado de arte de mecanismos de armazenamento de dados espaciais e as possibilidades de particionamento de dados georeferenciados. O objectivo principal do trabalho realizado foi definir um algoritmo para simular a política do GeoSharding e demonstrar como este pode optimizar o particionamento de dados georeferenciados utilizando um sistema de base de dados NoSQL.

Os resultados obtidos mostram que este método pode optimizar o particionamento da informação em relação ao desempenho das queries espaciais, do equilíbrio de carga nos nós do sistema e da replicação de dados em sistemas que armazenam informação espacial.

Palavras Chave

Sharding; Dados Georeferenciados; Big Data; NoSQL;

Contents

List of Figures

List of Tables

List of Acronyms

CAP	Consistency, Availability and Partition Tolerance
ACID	Atomicity, Consistency, Isolation and Durability
BASE	Basically available, Soft-state and Eventual consistency
RDBMS	Relational Database Management Systems
NoSQL	Not Only SQL
VON	Voronoi Overlay Network

1 Introduction

Contents

1.1 Motivation

Nowadays, there is data that can be georeferenced, which means that it can be associated with a physical space or location. This is a common term used in geographic information systems (further discussed in this thesis). One example of this type of georeferenced data is information associated to a determined geographic location that can be shared in a social network. Actually, social networks use several data like friends' groups, pictures or posts that can be georeferenced. The ever increasing observations and measurements of geo-sensor networks, satellite imageries, point clouds from laser scanning, geospatial data of Location Based Services (LBS) and location-based social networks has become a serious challenge for data management and systems' analysis [11].

The huge development of computer networks promotes data decentralization and distributed databases allow sharing this data making it accessible by all units and storing it where it is most frequently used. A distributed database is a logically interrelated collection of shared data, physically distributed over a network [12]. The data locality, the local where it is stored, must be conveniently choose in order to allow better data availability, access performance and other important characteristics further discussed in this dissertation.

Relational Database Model Systems are actually used in many scenarios where georeferenced data is stored, but there are some situations when using these systems may not provide the required efficiency and effectiveness. In these situations, NoSQL solutions can provide the efficiency necessary for applications using geospatial data. It is important to differentiate between the physical way a NoSQL product is implemented, and the interfaces, coding and access methods they use for the abstraction of data (in this case, spatial data) [13].

In addition, there are systems that process spatial data in a centralized manner and actual spatial partitioning mechanisms used during spatial data processing are not efficient (in the case of geohashing, or r-tree hashing, objects near each other geographically may end distributed by distant points in the space). Consequently, rises the problem that complex spatial queries do not take advantage of the georeferenced characteristics of data. Spatial range queries require the retrieval of data from a large number of different servers which is not efficient.

1.2 Proposed Solution

This work explores the georeferenced data characteristics to define a new data storage strategy. To solve the problem of efficiently store georeferenced data, one solution is to adopt a new partitioning policy/algorithm based on the geographical positions assigned to the objects to store, called GeoSharding. In the sharded database, each shard is assigned a geographical coordinate and it is responsible for storing the spatial objects "near" it.

GeoSharding allows optimizing the number of servers requested to solve a spatial range query which can possibly reduce network traffic and optimize data access. Furthermore, this new sharding policy also optimizes load balancing across the distributed database using it.

The mechanism proposed uses Voronoi Diagrams to perform spatial partitioning as it presents important advantages with respect to already in use spatial division mechanisms (geohashing, r-tree hashing). The irregular shape of Voronoi polygons optimizes the spatial indexing of irregular distributions of spatial objects which happens in real spatial environments.

1.3 Objectives and Results

The main goals of this master thesis are to prove that GeoSharding optimizes data partitioning in distributed databases that handle spatial data and to prove that this Voronoi solution presents a better data access performance and a better load balancing than the other spatial division mechanisms regarding the sharding mechanism proposed.

Various algorithms were tested in the experimental phase of this master thesis where its evaluation consisted in storing real spatial data sets and performing spatial queries in a simulated environment. Geosharding was compared to the actual sharding algorithms used to store spatial data sets.

The results presented show that this Sharding policy optimizes the number of servers requested to answer a spatial query in a distributed database environment and that GeoSharding also optimizes data load balancing across the different servers that compose the storage system using it.

1.4 Outline

The second chapter of this dissertation presents the Related Work required to understand the sharding strategy proposed. Chapter 3 illustrates GeoSharding and the details behind its implementation, followed by the Replication mechanism proposed. Chapter 4 presents the experimental tests performed to evaluated the proposed implemented mechanisms, being the results presented in Chapter 5. Finally, Chapter 6 presents the master thesis' conclusions.

2 Related Work

Contents

This chapter reviews the state of the art of Spatial Data, Data Partitioning, Geographic Data Storage Mechanisms and NoSQL Databases. The Spatial Data section presents the basics about geospatial data and operations. The Data partitioning section will review different partition techniques. The Geographic Data Storage Mechanisms and Algorithms section reviews some geographic indexing mechanisms and techniques. The NoSQL section will cover different motives of this systems' appearance, different system's classes and four already existing NoSQL databases, along with its respective partitioning algorithms.

2.1 Spatial Data

The study of the state of art of these geographic data systems gave the student the proper basis to solve the main problem of this master thesis, a sharding mechanism and partition algorithm based on the geographical location of the data.

In this context, it is important to have the notion of a spatial object and a spatial query. By the definition in [2], spatial objects are objects with extents in a multidimensional setting, and spatial queries are often executed by recursively subdividing the underlying space and then solving possibly simpler intersection problems. In terms of dimensional settings, this report focuses on systems, mechanisms and algorithms for bi-dimensional spaces.

A geographic/geographical information system or geographical information system (GIS) is a system designed to capture, store, manipulate, analyze, manage, and present all types of spatial or geographical data [14]. The actual GISs mainly store data in relational databases (RDBMSs) which makes the spatial data processing centralized (relying on relational mechanisms) and not distributed. Examples of RDBMSs used by GISs are MySQL or Oracle.

There is one international organization committed to making quality open standards for the global geospatial community, which is called OGS (Open Geospatial Consortium) [15]. These standards are made through a consensus process and are freely available for anyone to use in order to improve the sharing of the world's geospatial data.

The amount of georeferenced data is increasing everyday. There is a large variety of these data that goes from weather data, socioeconomic data, vegetation indexes, and more [16]. To understand the relevance of the work presented, it is important to mention some cases where manipulating and managing georeferenced Big Data is important.

When discussing geological phenomena, every occurrence of an event must be stored to further analysis or study. For example, the United States Geological Survey [17] is a scientific government agency responsible for the study of the landscape and the natural resources of the United States. This agency stores spatial data sets of earthquakes, water pits, volcanoes and landslides, among others, and delivers this information in real-time. Besides this agency, the British Geological Survey [18] is a natural environment research council that also works with large georeferenced data sets related to geological research. This includes climate change, groundwater, minerals, energy, among others.

Georeferenced data is also discussed when mentioning Social Networks, as the huge volume of data these applications have to manage and store is increasing everyday. The fact that nowadays users can pair a location dimension to the information they share in these type of networks adds georeferenced characteristics to this data volume and originates different location-based social networking services [19].

In the scope of the Internet of Things, once more is important to refer that geography can be considered an important binding principle, since all the sensor data gathered by all the physical objects have a position associated with, and there are spatial relationships between these objects. Consequently, there is a direct link between georeferences and Big Data in IoT since to spatially process the "data provenient from internet connected devices for real-time spatial decision making" [20], this data has to be georeferenced, creating "Spatial Big Data".

Regarding spatial data processing, there are several extensions and optimization applications that are being developed to optimize already existing technologies in order for them to work efficiently with georeferenced data. GeoSpar [16] is one example, and its role is to extend Apache Spark Resilient Distributed Datasets to support geometrical and spatial objects. This way, it is possible to create spatial indexes (discussed further in this section) that boost spatial data processing performance when performing data partitioning.

2.1.1 Spatial Data Types

The following two subsections are based in a background study [1], and cover an overview over spatial data types and common spatial operations.

The real world can only be depicted in a GIS through the use of models that define phenomena in a manner that computer systems can interpret, as well perform meaningful analysis. Spatial Data Models can be primarily classified into two different types, which determine how GIS data are structured, stored and processed, that are Vector Data Model and Raster Data Model.

The Vector Data Model uses points and their "x, y" coordinates to construct a spatial feature as a point, line, area and region. In this scope, a point may represent a well, a benchmark or a gravel pit whereas a line may represent a road, a stream or an administrative boundary. In fact, basic vector data can be divided into different types, which are presented below:

- Point Data: This is the simplest data type and it is stored as a pair of *X, Y* Coordinates. It represents the entity whose size is negligible as compared to the scale of the whole map such as it may represent a city or a town on the map of a country;

- Lines and Polylines: A Line is formed by joining two points in an end to end manner and it is represented as a pair of Point data types. In Spatial Databases, Polyline is represented as a sequence of lines such that the end point of a line is the starting point of next line in series. Polylines are used to represent spatial features such as Roads, Streets, Rivers or Routes, for example;

- Polygons: Polygon is one of the most widely used spatial data type. They capture two Dimensional spatial features such as Cities, States, and Countries etc. In Spatial Databases, polygons are represented by the ordered sequence of Coordinates of its Vertices, first and the last Coordinates being the same;

- Regions: Regions is another significant spatial data model. A Region is a collection of overlapping, non-overlapping or disjoint polygons. Regions are used to represent spatial features such as the

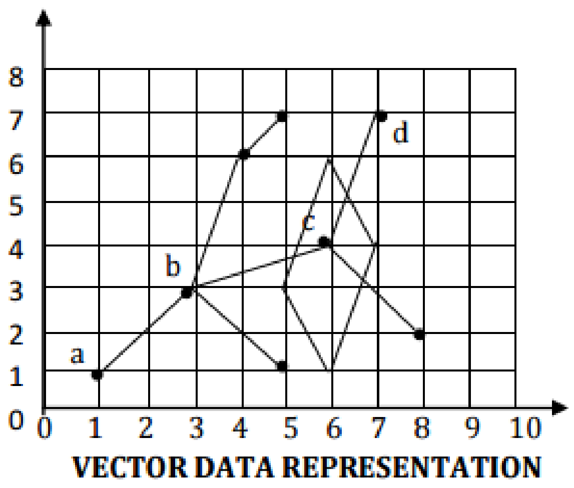


Figure 2.1: Polygon Representation [1]

State of Hawaii, including several islands (polygons), for example;

Figure 2.1 represents a polygon, which vertices belong to the following (x,y) coordinates: (6,1); (7,4); (6,6); (5,3); (6,1). Other name that can refer to these different types of spatial vector data is Geometry Objects.

The Raster Data Model exists to address the issue caused by the fact that vector data model does not work well with spatial phenomenon that vary continuously over the space such as precipitation, elevation and soil erosion. In fact, Raster Data Model uses Regular Grid to cover the space and the value in each grid cell to correspond to the characteristics of a spatial phenomenon at the cell location.

Conceptually, the variation of the spatial phenomenon is reflected by the changes in the cell value. A wide variety of used GIS are encoded in this format. They include digital elevation of data, satellite images, scanned maps and graphic files. Between the two Models, the majority of the spatial data stored in the web is represented by the Vector Data Model.

2.1.2 Common Spatial Operations

The geospatial information corresponding to a particular topic is gathered in a *theme*, which is similar to a relation as defined in the relational model and that also has a schema and instances [1]. A *theme* is hence a set of homogeneous geographic objects, and has two sets of attributes (Descriptive and Spatial).

There are several Geo-spatial operations applied to themes, which can represent a piece of land such as a state or a country. A Theme Union, where two themes having the same Descriptive attribute as well as spatial attribute types are combined together to form a larger theme of larger geometry, is one example of such geo-spatial operations. There are other examples like a Theme Overlay, where the Descriptive attributes of input themes are united whereas Geometry is intersected, or a Theme Selection, used to select only those tuples of theme which satisfies the given condition.

When discussing geospatial data, it is also important to mention Geocoding [21], which uses a description of a location to find geographic coordinates from spatial reference data such as building polygons or land parcels. This coding facilitates spatial analysis on GIS.

Other important aspects when working with spatial data access optimization are statistical methods that use geographical data correspondent to positions near each other. One example of such techniques is Kriging [22], an interpolation method used to predict a function value given a certain point, using the weighted average of the known values of the neighborhood of the selected point in the function. This is a very common method used in spatial analysis, and as many others take advantage of data locality.

The mentioned spatial operations over themes, like the Theme Union or the Theme Overlay, also take advantage of data locality since these operations work directly with the spatial component associated with the data composing the themes.

2.2 Data Partitioning

The volume of data to be stored in the web continues to grow more every year. Data in large scale systems exceeds the capacity of a single machine and should be partitioned and replicated, ensuring reliability and allowing scaling. This section of the Related Work will focus on data partition, partitioning types and other aspects regarding sharding.

2.2.1 Partitioning Types

There are various types of mechanisms that allow partitioning, depending on some systems' characteristics like dynamism or size [23]:

- Memory Caches:

  Memory Caches can be seen as partitioned in-memory databases because they replicate most frequently requested parts of a database to main memory. After, their job is to rapidly deliver this data to clients and disburden database servers significantly [24].

- Clustering:

  Clustering of database servers aims for transparency towards clients who should not notice talking to a cluster of database servers instead of a single server. One advantage of this approach is that it can help to scale the persistence layer of a system to a certain degree [23].

- Separating Reads from Writes:

  By specifying one or more dedicated servers, being the masters responsible for all the write operations, and the slaves by the replication (reading) operations, partitioning can be achieved [25].

  This master/slave model works well if the read/write ratio is high. The replication of data can happen either by transfer of state or by transfer of operations which are applied to the state on the slave nodes and have to arrive in the correct order.

- Sharding:

  This is the main partitioning strategy discussed along this dissertation and it is introduced in the next subsection.

2.2.2 Sharding

Sharding is the method of data partitioning in such a way that data typically requested and updated together resides on the same node and that load and storage volume is roughly even distributed among the servers (in relation to their storage volume and processing power) [23].

Data shards may also be replicated for reasons of reliability and load-balancing and it may be either allowed to write to a dedicated replica or to all the existing replicas, depending on the situations and maintaining a partition situation.

There must be a mapping between data partitions, the shards, and the storage nodes that are responsible for these shards in order to build a data sharding system. The client application decides if this mapping is static or dynamic by a direct contact or an interface network communication with the different nodes.

Sharding does not allow joins between data partitions, which means that the client application or the proxy layer inside or outside the database has to issue several requests and post process results instead. This results in the cancellation of some relational features of the RDBMS [26] like loosing the capacity to join data partitions, originated by the fact that sharding was not originally designed within current RDBMS but rather added on top.

In this way, many NoSQL databases embrace sharding as one key feature and some of them even provide automatic partitioning and balancing of data among nodes, which will be discussed further in the work.

2.2.3 Consistent Hashing

The mapping of the database's objects into servers is a fundamental key in a partitioned scenario. A common approach is using simple hashing of the database-objects' primary keys against the set of available database nodes in the following manner [23]:

$$partition = hash(o) \mod n,$$

with o being the object to hash and n being the number of network's nodes. One disadvantage of doing so is that some parts of data have to be redistributed whenever new nodes join or old ones leave.

In a memory caching scenario data redistribution may happen spontaneously by observing cache misses. However, this does not apply to persistent data stores as data not present on the available nodes cannot be reconstructed [25].

This problem may be solved by a different approach called Consistent Hashing, a family of caching protocols for distributed networks that can be used to decrease or eliminate the occurrence of hot spots in the networks [27].

This caching protocols family consists in a hashing algorithm that addresses both objects and caches using a hash function. To hash not only the objects but also the machines gives the advantage of giving the machines an interval of the hash-function's range, so adjacent machines can take over parts of the interval of their neighbors if those leave and can give parts of their own interval away if new nodes join and get mapped to an adjacent interval.

This solution also has the advantage of letting the client application calculate which node to contact in order to request or write a piece of data and there is no meta-data necessary as in systems which have a central meta-data server that contains the mappings between storage systems and data partitions (for example, Google File System [28]).

One example of a partitioning algorithm that uses Consistent hashing is Chord [29] (exemplified in Figure 2.2). Chord's main goal is to provide a "fast distributed computation of a hash function mapping keys to nodes responsible for them".

By using Consistent Hashing, Chord has high probabilities of balancing the load among the peers in the system (the nodes receive approximately the same volume of data).

In a typical Chord "ring", each node has an identifier value and is responsible for storing the objects whose key value is inferior than its own "id" value. Since the number of identifiers is limited, and the number of possible key values of the objects is also limited, the node with the highest identifier is linked to the node with the lowest identifier value, which is why the network forms the so called "ring". Figure 2.2 shows an example of a Chord Ring, where the node whose identifier is 5 can route messages to the nodes 6, 7, 9 and 21. Node 5 is responsible for storing objects whose key value belongs to the interval]2,5].

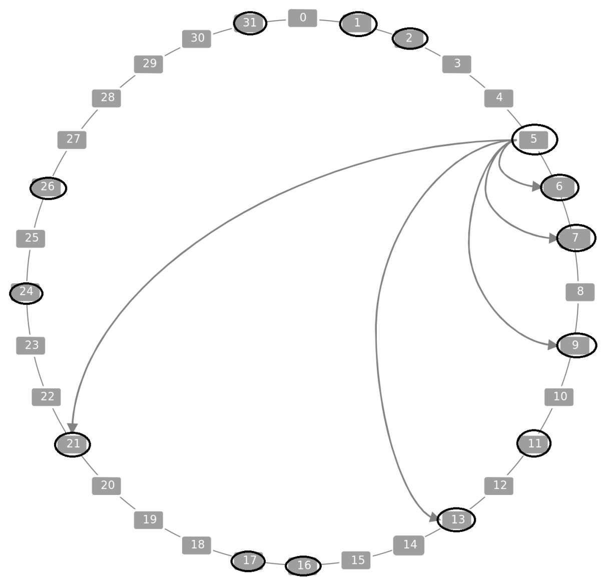


Figure 2.2: Chord Ring Example

Some advantages of using a Chord based algorithm to partition data in a distributed database are Load Balancing, Decentralization, Scalability and Availability. The importance of each of these four characteristics is described in section 2.4.

2.2.4 Replication and Read/Write Operations

In a partitioned scenario there is always the concept of data replicas, which besides bringing the advantage of reliability also make possible to spread the workload for read requests that can go to any physical node responsible for a requested piece of data [23]. This is relevant since a sharded database scenario requires balancing the workload between the available database servers. The Replication mechanism is one key to lower network communications when querying the database.

In fact, the possibility of load-balancing read operations does not apply to scenarios in which clients have to choose between multiple versions of a dataset and therefore have to read from a quorum of servers which reduces the ability to load-balance read requests.

In [5] it is stated that three parameters are specifically important when discussing read and write operations, which are the number of replicas for the piece of data to be read or written (N), the number of machines contacted in read operations (R) and the number of machines that need to be blocked in write operations (W).

It is also argued that in order to provide the consistency in the read-your-own-writes model, the following relation between the mentioned parameters becomes necessary:

$$R + W < N$$

In this case, it is guaranteed that a read will return the most up-to-date version (consistency). In terms of write operations, it can be assured to the clients that these operations are not immediately consistent or isolated, which means if a write operation completes without errors or exceptions a client can be sure that at least W nodes have executed the operation.

It also means that if the write operation fails in a way that less than W nodes have executed it, the state of the dataset is unspecified. If at least one node executed the operation, this node becomes the newest version of the correspondent replica nodes, as there is a communication between replica nodes in the background of a system with these characteristics. If the write operation fails completely, the client is forced to re-issuing the write operation again in order to achieve a consistent state.

Inside the scope of NoSQL databases it is significantly important to discuss the feature of the replication mechanism. In a database of this type it must always exist replicas of the objects stored, since these database systems need to be fault tolerant. By replicating the objects stored in the original shard owner, the shard's data availability is guaranteed even if the shard is temporarily down. Furthermore, the replication algorithms in NoSQL databases may allow data availability optimization, which will be further discussed in this master thesis.

2.3 Georeferenced data indexing

P2P applications require the capability to respond to more complex queries (such as range queries involving numerous data types, including those that have a spatial component) which results on studying different indexing strategies on data storage systems.

In order to proceed with this indexing in a bi-dimensional space, this systems use the longitude/latitude coordinates pair (x, y) to index data along the partitioned space. The purpose of this section is to describe and compare several spatial indexing mechanisms that can be used to implement the sharding mechanism proposed in this dissertation, where each shard has a position assigned: $(x, y) - > server$. By having a position assigned, each shard is responsible for storing the spatial objects inside a limited region of the space which is determined by the spatial data indexing mechanism used.

2.3.1 Geohashing

Geohashing is a latitude/longitude geocode system. It is a hierarchical spatial data structure which subdivides space into buckets of grid shape, and a convenient dimensional reduction mechanism for coordinates that uses a Z-Curve [30] to decide the visiting order of the partitioned quads, represented in Figure 2.3. A simple implementation is to simply interleave the bits of a (latitude, longitude) pair and *base32* encode the result.

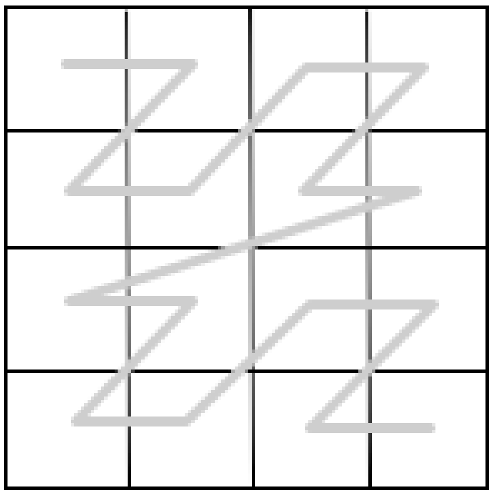


Figure 2.3: Geohashing Spatial Partitioning and Z-Curve Order

Geohashing features advantages such as facility on calculation and reverting, its representation in bounding boxes and the ability to truncate bits from the end of a Geohash resulting in a larger Geohash bounding the original [30]. Geohashes offer properties like arbitrary precision and the possibility of gradually removing characters from the end of the code to reduce its size (and gradually lose precision). As a consequence of the gradual precision degradation, nearby places will often (but not always) present similar prefixes. The longer a shared prefix is, the closer the two places are.

However, there are significant disadvantages in GeoHashing, also referred in [30], criticizing the Z-Curve form. Z-Curves are not necessarily the most efficient space-filling curve for range queries, since Points on either end of the Z's diagonal appear as close together when they are actually not.

Other inefficiency of the Z-curves is that points next to each other on the spherical earth may end up on opposite sides of the plane. These drawbacks mean that sometimes it is necessary to run multiple queries, or expand bounding box queries to cover very large areas.

2.3.2 CAN

CAN (Content Addressable Network) [31] is a P2P network protocol that uses consistent hashing based on bi-dimensional indexes. The CAN algorithm partitions space in rectangular blocks among all the nodes in the network. Every node "owns" a zone in the overall space and the system stores data at the networks' nodes. CAN allows routing from one "point" (a node that owns an enclosing zone) to another.

Space division in CAN is made in rectangular or quadrangular forms. In order for a new node to

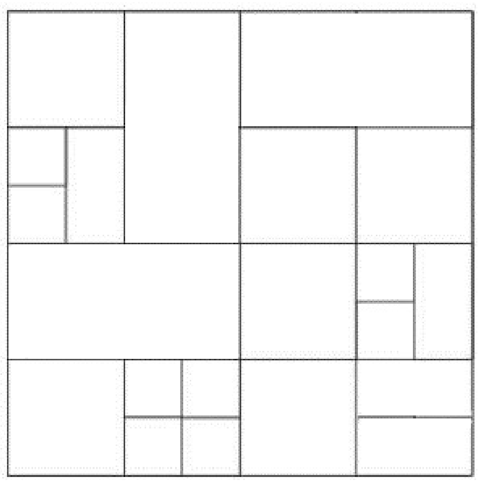


Figure 2.4: CAN's Spatial Division

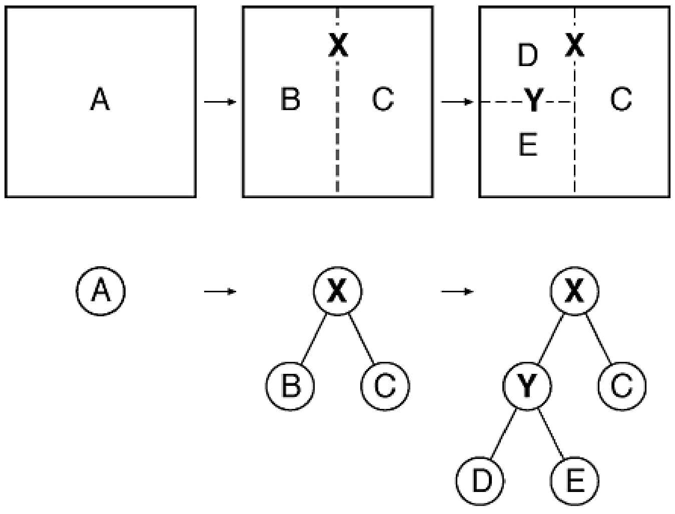


Figure 2.5: Content Addressable Network using a BSP Tree

join the network, the joining node has to perform the following steps [31]: 1) Find a node already in the overlay network; 2) Identify a zone that can be split; and 3) Join the network, updating the neighbors' information about the newly split zone.

In order to handle a departing node, CAN performs the following steps: 1) Identify the departing node; 2) Have the departing node's zone merged or taken over by a neighboring node; and 3) Update the routing information across the network.

Routing in CAN is made with the information of each node's neighbors. Every node maintains information about its direct neighbors and its virtual coordinate zone. A node routes a message towards a destination point in the coordination space. To do this, the node first determines which neighboring zone is closest to the destination point, and then looks up that zone's node's IP address using the network's information stored.

CAN is one of the spatial mechanisms evaluated further in this dissertation. Like the R-tree indexing [32], it performs spatial division using rectangular/quadrangular forms, exemplified in Figure 2.4.

CAN may be implemented using an Binary Spatial Partitioning Tree [33]. This data structure represents a recursive and hierarchical subdivision of n-dimensional space into convex sub-spaces.

Figure 2.5 exemplifies the BSP Tree, where there is 2-dimensional subdivision of space, as it is dealing with bi-dimensional spatial coordinates. Its construction process consists on taking a subspace and partition it by a line, parallel to the x-axis or the y-axis, that intersects the interior of that subspace, which results in two new sub-spaces that can be further partitioned by recursively applying the same method again.

One disadvantage of partition space in such a way is that it is not possible to choose the block's center location exactly, unless the space is sub-partitioned multiple times until a block is centered where it is desired. This is relevant in the scope of the sharding algorithm that will be introduced in the next chapter.

2.3.3 Quad-tree Hashing

Other indexing solution, presented in [2], also focused on the use of P2P networks for applications with spatial data and queries, is the use of a distributed spatial index that is based in the quad-tree data structure.

There are different existing variants of the quad-tree data structure, beginning with the region quad-tree (one of the most common). This first case consists on recursively decomposing an underlying two-dimensional square-shaped space into four congruent square blocks until each object is contained in any of the blocks.

One possible advantage of this first variant is that it reduces the complexity of the intersection process by enabling the pruning of certain objects or portions of objects from the query. Figure 2.6 represents the Quad-tree Hashing standard spatial partitioning.

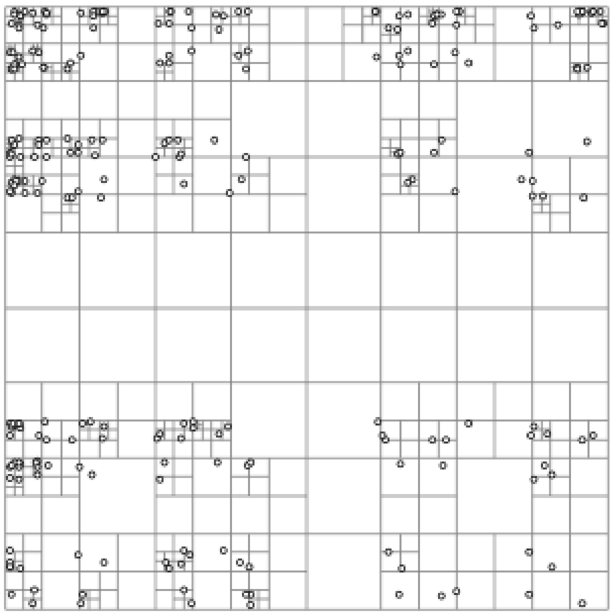
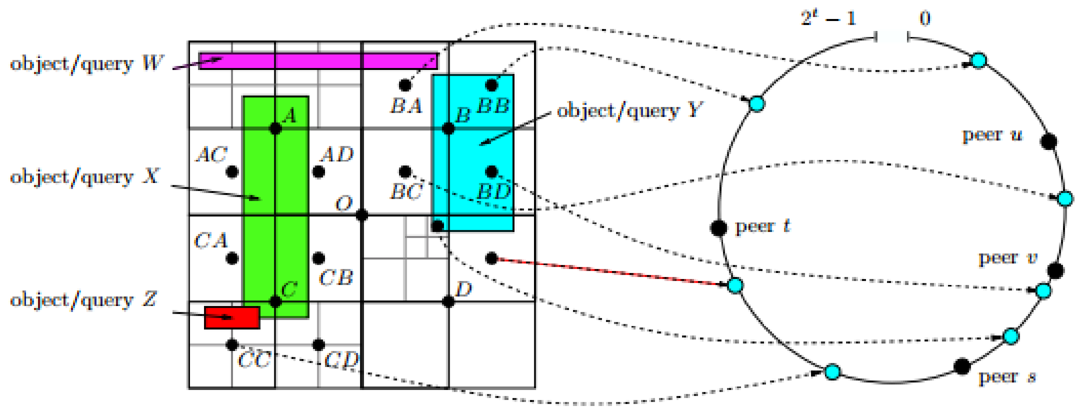


Figure 2.6: Quad-Tree hashing Spatial Division [2]

The MX-CIF quad-tree [2] is other alternative to implement a quad-tree index, that consists on: for each object *o*, the decomposition of the underlying space halts upon encountering a block *b* such that *o*



Spatial objects/queries $\{W, X, Y, Z\}$, control points, and some of the hashings to the Chord, i.e., the coordinate values of a control point are used as the key and hashed onto the Chord. Dark dots are the peers that are currently in the system. Light dots are the control points hashed on to the Chord. For this figure, $f_{min} = 2$.

Figure 2.7: Control points and hashings using Chord example [2]

overlaps at least two of the child blocks of b or upon reaching a maximum level of decomposition of the underlying space. In both the variants presented the object is associated with b upon halting.

When a peer is attached to a region of space it is responsible for all query computations that intersect that region, and for storing the objects that are associated with that same region. In order to do this, the authors declare that each quad-tree block can be uniquely identified by its *centroid*, named a control point. Posteriorly, it is used the Chord method to hash these control points so that the responsibility for a quad-tree block is associated with a peer.

To determine the control points, the system uses the globally known quad-tree subdivision method to recursively subdivide the underlying space. It is stated that multiple control points and hence quad-tree blocks can be hashed to the same peer and multiple objects can be stored with each control point.

The use of a control point in this algorithm [2] is analogous to that of a bucket for storing objects and also performing intersection calculations associated with that block.

To achieve an uniformly random mapping of the quad-tree blocks to peers of the network there is the necessity of using a good base hash function and in this case it is used SHA-1, a good candidate hash function used with Chord. This function maps two quad-tree blocks that are close to each other, as well as two peers with similar IP addresses, to totally different locations on the Chord space.

Posteriorly, as quad-tree subdivisions create new blocks for crowded regions of the space, the system will be assigning the responsibilities of these blocks to different peers, creating an implicitly load-balanced method that can handle skewed data distributions. Figure 2.7 presents some control points and hashing examples using the Chord method:

2.3.4 Voronoi Indexing

A Voronoi diagram is a division of a space into disjoint polygons where the nearest neighbor inside a polygon is the generator of the polygon [34]. The Voronoi diagram for 2-dimensional Euclidean space has many important properties described ahead in this section.

The formal definition of a Voronoi diagram can be given as follows: Considering a set of limited number of points (generator points), in the Euclidean plane, all the locations in the plane are associated to their closest generator. The set of locations assigned to each generator forms a region called Voronoi polygon or Voronoi Cell. The set of Voronoi polygons associated with all the generators is called the Voronoi diagram with respect to the generators set.

It is also declared [34] that the boundaries of the polygons, called Voronoi edges, are the set of locations that can be assigned to more than one generator (as these points are at the same distance from more than one generator). The Voronoi polygons that share the same edges are called adjacent polygons and their generators are called adjacent generators. Figure2.8 illustrates an example of a Voronoi diagram.

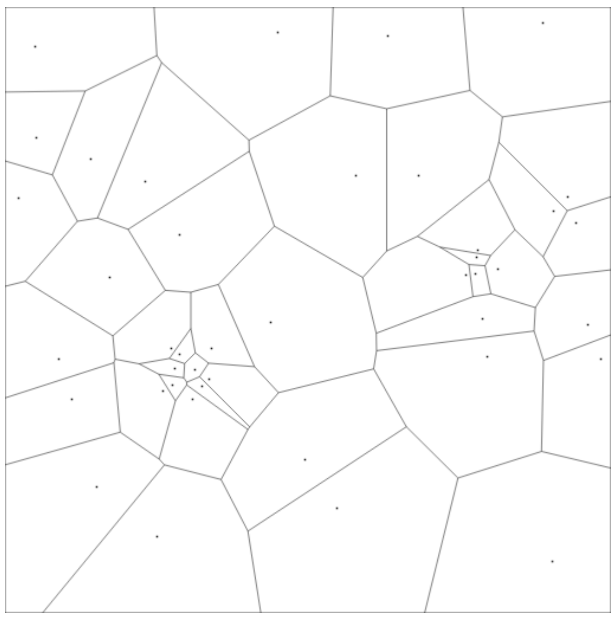


Figure 2.8: Voronoi Diagram's Example

There are several conclusions or solutions presented in different research works that are based in the important basic geometric Voronoi diagrams' properties, such as:

- The Voronoi diagram of a point set is unique;
- The nearest generator point of pi, for example pj, is among the generator points whose Voronoi polygons share similar Voronoi edges with pi's Polygon;
- Let n and ne be the number of generator points and Voronoi edges, respectively, then $ne \leq 3n–6$;
- From the previous property, and the fact that every Voronoi edge is shared by exactly two Voronoi polygons, we notice that the average number of Voronoi edges per Voronoi polygon is at most 6. This means that on average, each generator has 6 adjacent generators.

A network Voronoi diagram [34] is defined for graphs and is a specialization of Voronoi diagrams where the location of objects is restricted to the links that connect the nodes of the graph and distance between objects is defined as the length of the shortest distance in the network instead of their Euclidean distance.

Spatial networks can be modeled as weighted graphs where the intersections are represented by the links connecting the nodes. The weights can be the distances of the nodes or they can be the time it takes to travel from a node to another.

2.3.4.A VAST

VAST [3,35] is an open source Voronoi Overlay Network, whose description and features are presented in this section.

VAST was developed to answer the need of a highly scalable and affordable multi-user NVE (networked virtual environment), inside the scope of online gaming issues. This new NVE would have to allow millions of users to play at the same time using a peer-to-peer computing approach, being the peer-to-peer network used called Voronoi-based Overlay Network. The work developed was focused on the problems of scalability, to answer the need of allowing a large number of concurrent users in a NVE system, letting aside other issues like security, reliability and consistency.

To describe this network, it will be assumed the scenario where each peer in the network is called node, and each node's "visibility"(the node's knowledge about the other nodes in the network) is called Area of Interest (AOI), which is represented by a circle centered in the peer. Although the network is composed by several nodes, each one of these only is aware of the neighbors inside its AOI. Figure 2.9 presents a node and its area of interest (blue color).

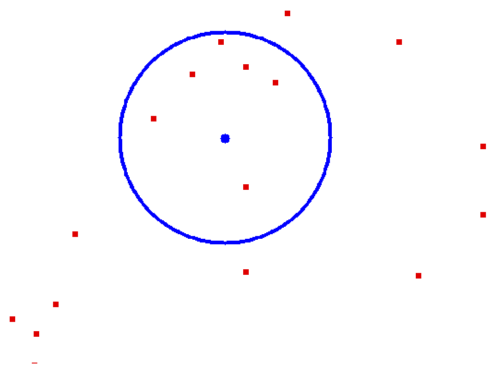


Figure 2.9: Node's Area Of Interest [3]

In a NVE scenario, each node is typically more interested in what is happening in the nodes closer to it, being the relevant messages belonging to its AOI neighbors, which turns the main problem to "finding the proper AOI and its neighbors" as other nodes move, fail or join. This is treated like a neighbor discovery problem.

The approach behind VAST is that each node only connects and trades messages with its Area of Interest neighbors. To do this, each node will build a Voronoi diagram with its coordinates and all its AOI neighbors' coordinates. In order to perform neighbor discovery, the nodes communicate with their boundary neighbors (the neighbors whose Voronoi region overlap with the node's area of interest circle boundary). This happens because these boundary neighbors may know what other nodes exist in the network outside the AOI.

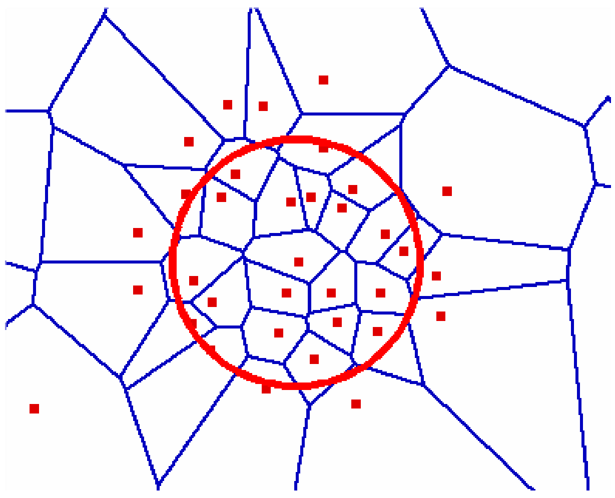


Figure 2.10: Crowding Situation Example [3]

To implement the updating requirements, a node sends updates to all its AOI neighbors when it moves. When receiving this updates, the boundary neighbors have to check if there are new potential neighbors to the node that is moving, as its area of interest may now include new peers. To exclude the case where a node is disconnected of the network because it has no neighbors in its area of interest, each node has to maintain connections to its enclosing neighbors (the neighbors determined by the Voronoi regions that surround a node's Voronoi region) [3].

As stated by the authors [35], Voronoi Overlay Network (VON) (Voronoi-based Overlay Network) meets the requirements of message-efficiency, as the neighbor discovery is embedded in regular position updates, scalability, as each node only maintains a limited number of connections and responsiveness, as latency is low due to the direct connections between neighbors.

In the situations of crowding (when there are several nodes closer to each other, like presented in Figure2.10 there is the possibility of message overload. To prevent this, VAST implements a dynamic adjustment of the Area of Interest, based on the following steps [3]:

1. Area of Interest's radius (the region's circle radius) diminishes when a certain "number of connections exceeds a predefined connection limit";

2. AOI's radius enlarges to its original state when the number of connections of the node decreases below the connection limit;

3. The nodes maintain a state of mutual awareness, which means that if node A cannot see node B due to overload reasons, node B has also to diminish its area of interest to not include node A.

2.3.4.B ToSS-it

The following subsection presents other indexing mechanism based in Voronoi diagrams. The work presented in [36] consists in an index for moving objects that support queries efficiently and also cope with frequent updates named ToSS-it. It is relevant to explore a Voronoi based mechanism for moving objects since it will be also discussed a similar mechanism further in the work.

The main idea behind this approach is to first distribute all point objects across multiple cloud servers, and then build local Voronoi diagrams at each server. The authors in [36] discuss their approach in order to minimize the number of messages passed between the cloud servers during the Voronoi construction. It is declared that ToSS-it can be an alternative to RT-CAN, a routing protocol based on CAN.

Behind the ToSS-it approach is an ideal of distribute-first-build-later. For this to happen, after effectively distributing the objects across the nodes (balancing the load of each node) it is necessary to build the local Voronoi diagrams at each node.

The challenge that rises is that even though the objects are distributed effectively, when the set of scattered point objects O is divided into disjoint subsets, some of the generated local Voronoi diagrams will be inaccurate as their neighbors reside in different nodes. The nodes need to communicate between each other to validate the accuracy of Voronoi cells and fix them if needed, which yields significant network I/O. To address this challenge, ToSS-it has three important steps/mechanisms:

1. Data Partitioning: The ideal partitioning should preserve spatial locality among the objects in each partition to minimize the number of inaccurate Voronoi cells (and hence message passing between the nodes). Towards this end, the authors [36] propose a Voronoi-based partition technique, based on the main idea that after selecting a number of pivot data objects (where each pivot corresponds to a partition) it is assigned each object to its closes pivot.

2. Local Voronoi Diagram Generation: In this phase, each node is asked to generate a local Voronoi diagram for the corresponding objects.

   There are two specific goals in this phase: to aim to exploit all multi-core CPUs available at the nodes to generate local Voronoi diagrams and effectively identify and fix Voronoi cells which might be inaccurate because of partitioning. To identify inaccurate cells, it is started to compute, for each object, an influence region and then check this region against the borders of the principal node. If the object's influence region falls inside the node, the object does not need to be redefined. Otherwise, the object is marked as inaccurate.

   In fact, another important point is that these inaccuracies can be identified locally without information about other nodes, which allows this system to provide high scalability, because every server first detects and then fixes the inaccuracies in parallel.

3. Geospatial Replication: After declaring that local Voronoi diagrams in each node include inaccurate cells, and that the inaccurate cells can be detected using the Influence Region, it is stated [36] that the generator (data object) of those identified cells must be replicated across the partitions.

   This way, the potential inaccurate Voronoi cells are redefined using local information. This step allows avoiding a significant amount of network communication.

After, it is stated that query processing can be improved by building local hierarchies on top of local Voronoi diagrams. It is defined a Voronoi hierarchy as follows: A Voronoi cell of the higher level-i contains all Voronoi generators of the lower level-i-1 that are closer to that level-i generator than to any other.

The main advantage of this approach is that Voronoi hierarchies can be built by exploiting multi-core CPUs efficiently.

2.4 NoSQL Databases

The following subsections will describe the NoSQL databases' development motives and some of the existing technologies of this type. There will be also be discussed the CAP-theorem, an important aspect regarding data storage systems. Then, the main focus will be on the respective databases' partition algorithms.

As each existing NoSQL database presents a unique architecture, it will be described and illustrated some of the already existing NoSQL systems. Three of the databases presented were chosen because they belong to three companies that have to deal with a huge amount of data, each one dealing with different challenges and needs brought by its millions of users (Dynamo, Cassandra and Bigtable). MongoDB is presented in this thesis because it is also a database that already deals with georeferenced data (this important detail will be further discussed).

As Dynamo and Bigtable are not open source systems, two similar alternative systems are, respectively, Project Voldemort (Dynamo) and Hypertable or HBase (Bigtable). For each presented NoSQL system, it will be described its partitioning/sharding algorithm, since the optimization strategy proposed in this thesis is related to NoSQL systems' partitioning techniques.

Nowadays, the predominant systems used to store structured data in web and business applications are the Relational Database Management Systems Relational Database Management Systems (RDBMS)s which rely on the relational calculus and providing comprehensive ad-hoc querying facilities by SQL [23].In the past, these systems have been treated as the only reliable way to store data by many companies and their respective clients [37].

However, in the last few years, the "one size fits all" idea has been questioned by science and web affine companies, which resulted in the appearance of a great variety of alternatives to the RDBMSs. These new datastores technologies which rely on non-relational databases may be called NoSQL (Not Only SQL) databases [23].

2.4.1 Motives of NoSQL practioners

Recently, NoSQL users came to share how they had surpassed the slow and expensive systems based on RDBMs [38]. Some of the main motives and advantages of NoSQL databases are presented here [23]:

Relational systems present some variety features to ensure data consistency that sometimes are unnecessary. For example, for some clients or some specific application there are some consistency checks that could be avoided and which are responsible for a larger complexity of the system. Therefore, one advantage NoSQL systems may provide is the **avoidance of unneeded complexity**.

Many times, in big data storage systems, it is important to increase the data processing rate in order to satisfy users' needs. Taking as an example Google's Bigtable [39] (Google's storage system), it is known that Google is able to process 20 petabyte per day via its MapReduce approach, which is a higher processing throughput than a traditional relational system. So, other motive of an alternative system's usage can be the **high throughput**.

As the volume of data increases with time, the focus on scaling and sharding increases too. In contrast to relational database management systems, most NoSQL databases are designed to scale well in the horizontal direction and not rely on highly available hardware. **Running on commodity hardware and presenting horizontal scalability** can be considered as other of the main drivers to develop this type of technologies.

Some types of data structures that present low complexity, when stored by RDBMS, require an expensive entity-relational mapping, which can be unnecessary. The **avoidance of expensive object-relational mapping** is also important [40].

Sometimes, NoSQL systems can also **compromise reliability for better performance**. This is important because, once again, the need of a higher system's speed may overcome the reliability's need [41].

As it was described, the continuous growth of data volumes to be stored and the growing need to process larger amounts of data in shorter time motivated the appearance and creation of NoSQL databases. The scalability of these systems allows balancing the network's data load, which results in a better performance of data availability.

2.4.2 The CAP-Theorem

It is important to discuss some important characteristics/requirements of shared data systems, like Consistency, Availability and Partition Tolerance. These three characteristics form the acronym Consistency, Availability and Partition Tolerance (CAP), and they can be described as [8]:

Consistency: Meaning if and how a system is in a consistent state after the execution of an operation. A distributed system is typically considered to be consistent if after an update operation of some writer all readers see his updates in some shared data source (even so, there are other alternatives for the definition of consistency).

Availability or High Availability: Meaning that a system is designed and implemented in a way that allows it to continue operation (allowing read or write operations) even if nodes in a cluster crash or some hardware or software parts are down to upgrades.

Partition Tolerance: Understood as the ability of the system to continue to operate in the presence of network partitions. These occur if two or more "islands" of network's nodes cannot connect each other (temporarily or permanently). Some people also understand partition tolerance as the ability of a system to cope with the dynamic addition and removal of nodes.

The three characteristics above mentioned may present different levels of importance, depending on the main goals of the storage system discussed. However, the CAP theorem states that one partition storage system may only have, at most, two of those three characteristics.

One possible solution (choice) to the theorem for a system that needs to be consistent and partition tolerant is to build a system according to "Atomicity, Consistency, Isolation and Durability (ACID)" (Atomicity, Consistency, Isolation, Durability), forfeiting the availability characteristic. By giving up the

Table 2.1: CAP Theorem – Alternatives, Traits and Examples by [8]

Choice	Traits	Examples
Consistency + Availability	2-phase-commit Cache-validation protocols	Single-site databases Cluster databases LDAP xFS file system
Consistency + Partition tolerance	Pessimistic locking Make minority partitions unavailable	Distributed databases Distributed locking Majority protocol
Availability + Partition Tolerance	Expirations/leases Conflict resolution Optimistic	Coda Web caching DNS

consistency, one can reach the "Basically available, Soft-state and Eventual consistency (BASE)" solution properties (Basically available, Soft-state, Eventual consistency), useful for systems that need a higher performance.

In Table2.1 there are some traits and examples of the three main possible choices for the CAP theorem [8]:

Referring other examples, it can be seen in Amazon's Dynamo [9] a system available and partition-tolerant but not strictly consistent (BASE properties). On other hand, Google's Bigtable [39] is fully consistent and available although it does not fully operate in the presence of network partitions.

Some of the actual most relevant Not Only SQL (NoSQL) database systems will be presented in the next section, along with some of their characteristics and requirements.

2.4.3 Facebook's Cassandra

Facebook runs the largest social networking platform that serves hundreds of millions users, using tens of thousands of servers located in many data centers around the world. Facebook has to deal with high scalability, performance, availability, efficiency and reliability requirements [42].

In fact, Cassandra was designed to solve the Inbox Search problem, which consisted on allowing the users to search through their Facebook Inbox, which means dealing with a very high write throughput (billions of writes per day) and also scale with the number of users. As the key to lower the latency, Cassandra is able to replicate data across data centers, since users are served from data centers that are geographically distributed.

Referring to its data model, a table in Cassandra is a distributed multi-dimensional map indexed by a key and the value is an object which is highly structured. Every operation under a single row key is atomic per replica no matter how many columns are being read or written to.

It is also explained [42] that in order to support applications and features like the Inbox Search application, Cassandra presents two kinds of column families, Simple and Super column families. Super column families can be visualized as a column family within a column family.

Regarding to its distributed system technique, Cassandra relies on partitioning, replication, membership, failure handling and scaling to fulfill its needs and requirements, such as handling the read/write

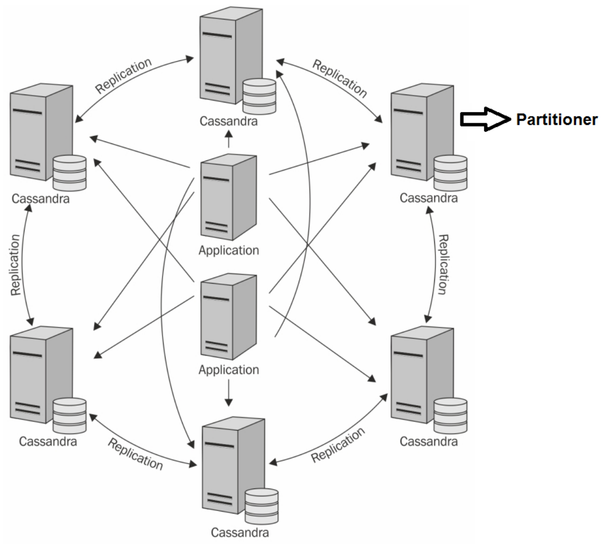


Figure 2.11: Cassandra's arquitecture [4]

requests.

Typically a read/write request for a key gets routed to any node in the Cassandra cluster, and then the node determines the replicas for this particular key. For writes, the system routes the requests to the replicas and waits for a quorum of replicas to acknowledge the completion of writes. For reads, based on the consistency guarantees that the clients need to see fulfilled, the system either routes the requests to the closest replica or routes the requests to all replicas and waits for a quorum of responses [42].

Cassandra's arquitecture is also a decentralized arquitecture [23]. To achieve high scalability and durability, each of Cassandra's cluster has a partitioner, which determines how to distribute the data across the nodes in the cluster and which node to place the first copy of data on. Basically, a partitioner is a hash function for computing the token of a partition key. Each row of data is uniquely identified by a partition key and distributed across the cluster by the value of the token. Figure2.11 illustrates Cassandra's architecture.

Facebook's datastore system Cassandra has the ability to scale incrementally. This ability requires the dynamic partition of the data over the set of nodes (storage hosts) in the cluster. Cassandra partitions data across the cluster using consistent hashing but uses an order preserving hash function to execute this technique. In fact, in this case the output range of a hash function is treated as a fixed circular space (the largest hash value wraps around to the smallest hash value), where each node in the system is assigned a random value within this space which represents its position on the "ring".

Each data item identified by a key is assigned to a node by hashing the data item's key to yield its position on the ring and then walking the ring clockwise to find the first node with a position larger than the item's position [42]. That node is deemed the coordinator for this key, that same key is specified by the application and the Cassandra uses it to route requests.

Each node becomes responsible for its region, which corresponds to the area between it and its predecessor in the ring. One advantage of consistent hashing is that the arrival or the departure of one node only affects the nodes in the regions near the departed or recently arrived node, remaining all the other nodes unaffected.

However, this type of hashing presents some complex challenges, like the random position assignment to nodes leading to non-uniform data and load distribution. Besides that, the basic consistent hashing algorithm ignores the heterogeneity in the performance of the nodes.

Two possible solutions for these problems are assigning the nodes to multiple positions in the circle or analyze load information on the ring and have lightly loaded nodes move on the ring to lighten heavily loaded nodes. The system uses the second solution presented, since it makes the design and implementation very tractable and helps to make very deterministic choices about load balancing.

In what matters to membership [42], cluster membership in Cassandra is based on Scuttlebutt, a very efficient anti-entropy Gossip based mechanism that has very efficient CPU utilization and very efficient utilization of the Gossip channel. Within the Cassandra system, Gossip is not only used for membership but also to disseminate other systems related control state.

Cassandra does not have direct support for spatial queries or spatial indexing, being necessary proper extensions for this NoSQL system to work with this type of indexing.

2.4.4 Amazon's Dynamo

Amazon is a company responsible for running a world-wide e-commerce platform, which has the purpose to serve tens of millions of customers using tens of thousands servers located in many data centers around the world [9]. It is mentioned that, as it could be expected, there are strict operational requirements on the Amazon's platform in terms of performance, reliability and efficiency, and to support continuous growth the platform needs to be highly scalable. In addition, reliability is other very important requirement for this platform.

Dynamo is a highly scalable and reliable storage system that Amazon has developed in order to meet such requirements, and it needs tight control over the trade-offs between availability, consistency, cost-effectiveness and performance. As a Key-/Value Stores system type, Dynamo provides a simple primary-key only interface to meet the requirements of some of the Amazon services that only need primary key access to a data store, avoiding lack of performance or unneeded complexity.

To achieve scalability and availability, this system uses some techniques such as consistent hashing to partition and replicate the data (this technique will be explained further in the dissertation). The consistency among replicas during updates is maintained by a decentralized replica synchronization protocol. The system also uses a gossip based distributed failure detection and membership protocol.

One big advantage of this NoSQL system is that it is completely decentralized, with minimal need for manual administration. Storage nodes can be added and removed from Dynamo without requiring any manual partitioning and redistribution [9]. Table 2.2 presents a summary of the techniques used in Dynamo and their respective advantages.

As mentioned before, in Dynamo's arquitecture, all nodes have equal responsibilities, there are no distinguished nodes having special roles. This architecture favors "decentralized peer-o-peer techniques over centralized control". Storage hosts added to the system can have heterogeneous hardware which Dynamo has to consider to distribute work proportionally "to the capabilities of the individual servers"

Table 2.2: Summary of techniques used in Dynamo [9]

Problem	**Technique**	**Advantage**
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background
Membership and failure detection	Gossip-based membership protocol and failure detection	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information

[23].

Since Dynamo is not an open source code system, Figure 2.12 presents Project Voldemort's [5] architecture (the open source version of Dynamo), where "Load Bal." indicates a hardware loadbalancer or round-robin software load balancer and "Partition-aware routing" is the storage system's internal routing. In this system there is flexibility on where the intelligent routing of data to partitions is done. This can be done on the client side for "smart" clients, or it can be done on the server side to enable "dumb", hardware load-balanced *http* clients.

As already mentioned and like Facebook's data storage system, Amazon's Dynamo has the requirement of scaling incrementally. This system also uses the consistent hashing mechanism to achieve its partition goals.

To solve the consistent hashing challenges presented in section 2.4.3, instead of mapping a node to a single point in the circle, each node gets assigned to multiple points in the ring.

To this end, Dynamo uses the concept of "virtual nodes", a node that looks like a single node in the system but each node can be responsible for more than one virtual node.

Effectively, when a new node is added to the system, it is assigned multiple positions, called "tokens" in the ring. Using these virtual nodes has the following advantages:

1. If a node becomes unavailable (due to routine maintenance or failures), the load handled by this node is evenly dispersed across the remaining available nodes;

2. When a node becomes available again, or a new node joins the "ring", the newly available node accepts a roughly equivalent amount of load from each of the other available nodes;

3. The number of virtual nodes that a node can be responsible for may be decided based on its capacity, accounting for heterogeneity in the physical infrastructure.

Regarding spatial data indexing, Dynamo supports geohashing, a technique described already. This database also supports two types of spatial queries: Bounding Box queries and radius queries.

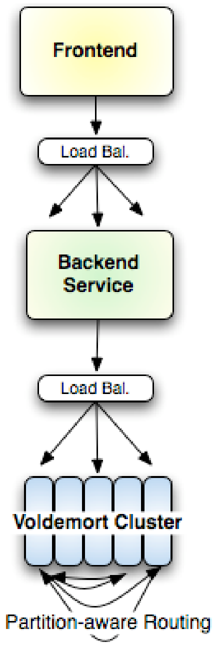


Figure 2.12: Project Voldemort's physical arquitecture [5]

2.4.5 Google's Bigtable

Like both systems presented before, Google has the necessity of having a data storage system that can scale to a very large size. Bigtable is designed to reliably scale to petabytes of data and thousands of machines [39].

Google has many applications which functionality depends on the Bigtable's system and need several characteristics like scalability, high performance and high availability. These same applications use Bigtable for a variety of demanding workload, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users.

Google's team also refers that Bigtable differs from normal parallel or main-memory databases in its interface, providing clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage.

In fact, data is indexed using row and column names and is treated as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can also control the locality of their data through careful choices in their schemes. An important aspect is that this system's schema parameters let clients dynamically control whether to serve data out from memory or from disk.

It is also referred that besides the performance and high availability provided by this system, the users like the fact that Bigtable can scale the capacity of their clusters by simply adding more machines to the system as their resource demands changes over time.

Bigtable's architecture is built on several other pieces of Google infrastructure and it uses the distributed Google File System to store and log files. A Bigtable cluster typically operates in a shared pool of machines that run a wide variety of other distributed applications, and Bigtable processes often share

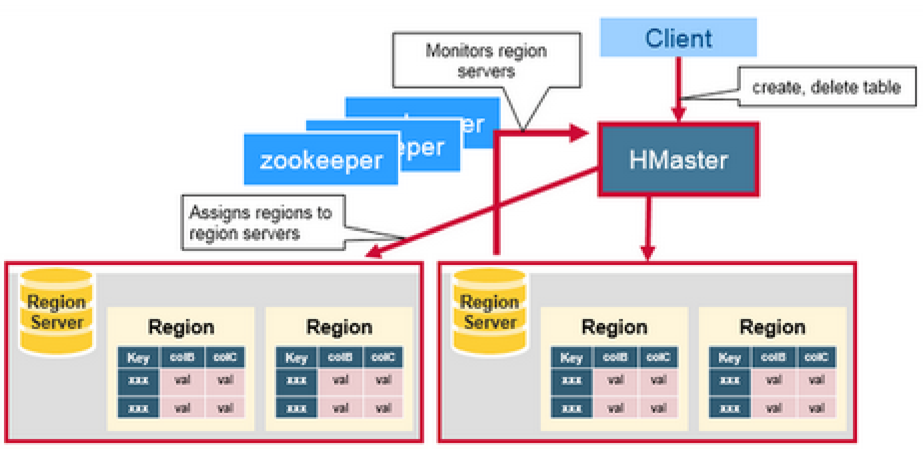


Figure 2.13: HBase's arquitecture [6]

the same machines with processes from other applications. This system depends on a cluster management system for scheduling jobs, managing resources on shared machines, dealing with machine failures and monitoring machine status. As it is also not an open source system, Figure 2.13 presents the architecture of HBase, one similar open source solution, also centralized.

In order to understand Google's data storage system's partitioning, it is important to refer that this partitions happen in a centralized environment. Bigtable relies on a highly-available and persistent distributed lock service called Chubby. A Chubby service consists of five active replicas, one of which is elected to be the master and actively serve requests [39].

The master is responsible for assigning tablets to tablet servers, detecting the addition and expiration of a tablet servers, balancing tablet-server load, and garbage collection of files in the Google File System. In addition, it handles schema changes such as table and column family creations.

Each tablet server manages a set of tablets (typically there can be between ten to a thousand tablets per tablet server). The tablet server handles read and write requests to the tablets that it has loaded, and also splits tablets that have grown too large.

As with other single-master distributed storage systems, client data does not move through the master: clients communicate directly with tablet servers for reads and writes. Since Bigtable clients do not rely on the master for tablet location information, most clients never communicate with the master. This results in lightly loaded masters.

Each Bigtable cluster store a set of tablets, and each tablet contains all data associated with row range. Initially, each table consists of just one tablet. As a table grows, it is automatically split into multiple tablets [39].

Bigtable uses Chubby to keep track of tablet servers. When a tablet server starts, it creates, and acquires an exclusive lock on, a uniquely-named file in a specific Chubby directory. The master monitors this directory to discover tablet servers. A tablet server stops serving its tablets if it loses its exclusive lock, which can happen in case of a network partition that caused the server to lose its Chubby session. Whenever a tablet server terminates, it attempts to release its lock so that the master will reassign its tablets more quickly.

Originally, Bigtable does not have direct support for spatial queries or spatial indexing, being necessary proper extensions for this NoSQL system to work with this type of indexing.

2.4.6 MongoDB

MongoDB is a document store open source NoSQL database system, used by the social network called Foursquare [43]. Its creation had the purpose of facilitating scaling using its Sharded Cluster architecture, which also presents high availability and fault tolerance.

MongoDB's inner structure is composed by collections inside databases (being the collections the equivalent to the relational model tables) [7]. The Sharded Cluster architecture is composed by nodes which contain three configuration processes (config servers), one or more replica sets and routers. Each of these components has different responsibilities in the whole cluster system. Figure 2.14 illustrates MongoDB's architecture:

In this kind of architecture, a replica set is a MongoDB's instance cluster which implements a master-slave replication mechanism between them. The instances also implement automatic failover mechanisms between them.

The router has the functions of forward requests, process application requests, determining where to store the information inside the cluster and also process load balancing. The router is also the interface between the application and the whole cluster.

A shard can be a simple "mongod" instance or a replica set and it stores a share of the cluster's information. Config servers are also "mongod" instances that store the Sharded clusters' meta-data. In fact, in a production environment it is recommended that a Sharded Cluster architecture presents two router instances ("mongos"), three config server instances ("mongod") and at least two replicas set configured Shards.

In a cluster's router configuration, in this type of arquitecture, there is a parameter called *chunkSize* that defines the document group size to be stored in a Shard, in order to achieve the limit that allows a split operation between Shards [7].

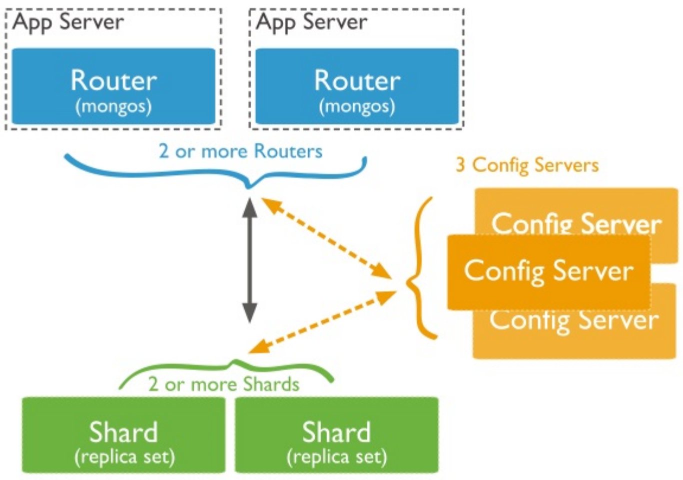


Figure 2.14: MongoDB's arquitecture [7]

It is also stated that the distribution made between shards happens at the collection's level, and is achieved using a Shard Key, a simple or complex field that exists in every document stored in a collection. To split a Shard key there are two different mechanisms, a value based partitioning or using hashing. These two partitioning strategies will be discussed further in this thesis.

When a new Shard is inserted in the architecture, the system's data is redistributed automatically considering already the new existing Shard. If there is a big volume of data already stored in the system, this redistribution may take more time than what is pretended, as there is a big data volume traffic going in the network between the Shards.

One possible solution to this problem is the creation of virtual Shards, which are configured in the same physical node. By using these virtual shards, the data redistribution happens in the same physic node [7].

Regarding spatial data indexing, MongoDB supports both geohashing and quad-tree hashing, that were already presented. This database also supports spatial queries like Bounding Box queries and radius queries.

2.5 Summary

One of the topics reviewed in this chapter was spatial data, which is continuously growing in the web. This type of data is commonly stored in relational databases, used by the GIS systems. To address the challenges of storing huge volumes of data, there are actual databases that work with data partitioning to achieve scalability and more efficient data storage.

There are already indexing mechanisms that work with georeferenced data and some of these georeferenced data indexes are already used by NoSQL systems, databases that work with data partitioning to store data. Besides the databases presented, there are other NoSQL systems that work with spatial data like Lucene and IBM Cloudant [10].

To conclude the chapter of Related Work, Table2.3 presents a list of NoSQL systems that already work with some of the spatial indexing mechanisms presented along this chapter and the spatial query and data types supported by those same systems.

Table 2.3: Index strategy, Data type and Query Type of different data storage systems [10]

Database	**Index strategy**	**Data types**	**Query Types**
Dynamo	GeoHashing	point	BBOX, Radius
Cassandra	-	-	-
IBM Cloudant	R-Tree Hashing	GeoJSON types	BBOX, Radius, Arbitrary Shapes
Lucene/Solr	GeoHashing	Point	BBOX, Radius
Bigtable	-	-	-
MongoDB	GeoHashing/Quad-tree	GeoJSON types	BBOX, Radius, Arbitrary Shape

3 GeoSharding

Contents

Nowadays, georeferenced data volume on the web is continuously growing and the need of storing and accessing this data in efficient ways is a top priority regarding network technologies. The data locality redefinition (storing data in servers "near" it) requires a new solution that explores georeferenced data to allow a better system performance and efficiency.

As mentioned before, the index strategies are a very important topic regarding spatial networks. Now, these indexes can be used to efficiently access data on P2P networks, but the actual NoSQL databases do not use spatial indexing strategies efficiently.

The first goal of the sharding strategy proposed in this thesis is to optimize spatial partitioning in spatial indexing mechanisms. The second objective is to optimize spatial range queries and the load balancing with the proposed sharding strategy.

GeoSharding is the sharding algorithm proposed that consists on using Voronoi Spatial Partitioning as spatial indexing/sharding mechanism. Each database's shard is attributed a coordinate in the space and using the object's georeference as sharding key, the data is stored in the "closest" server. The coordinates attributed to each server can be random, chosen, or even dynamic, so that is possible to optimize data load balancing. Since the idea of using the spatial coordinates as the sharding key could also be implemented using other spatial mechanisms (CAN, Geohashing, Quad-tree Hashing), the work developed will compare this regular spatial mechanisms with Voronoi diagrams and show why the proposed spatial mechanism is the optimal solution.

This chapter explains how Voronoi indexing can be used, the Paradox between Range Queries and Load Balancing regarding Sharding, the algorithms' design proposed, the spherical vast usage and the replication mechanism proposed.

3.1 Spatial Partitioning

Current geo-indexing mechanisms (such as geohashing, quad-tree or CAN) present some disadvantages that may affect the spatial indexing system's efficiency. Table 3.1 presents the overview of the possible spatial indexing strategies(presented in the previous chapter) to be used with NoSQL systems, its partition block shape, the partition center location of each block (the central node responsible for each spatial block) and the partition depth (if sub partitions are allowed) of each mechanism. The Ruled partition center location refers to the centers of the rectangles/squares used by an index to partition space.

Table 3.1: Spatial Index Strategies

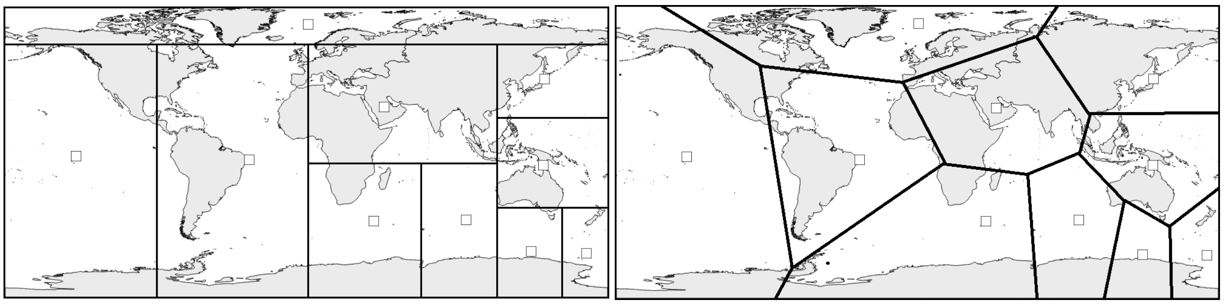
Indexing	Partition Blocks	Partition Center Location	Partition Depth
GeoHashing	Regular Rectangle	Regular	Regular
Quad-Tree Hashing	Irregular Square	Ruled	Sub-partitions
R-Tree Hashing (CAN)	Irregular Rectangle	Ruled	Sub-partitions
VAST	Irregular Polygon	Random	Irregular

As mentioned before, the Z-Curves of geohashing are not necessarily the most efficient space-filling curve for range queries, since Points on either end of the Z's diagonal seem close together when they are actually not. Besides range queries, nearest neighbor discovery using Z-curves may also be an inefficient solution which presents another problem in using geohashing.

The rectangular and quadrangular forms of the Geohashing, quad-tree hashing and CAN may also present inefficiencies. These systems' scalability obligates space subdivision which may not be, once again, an optimal way to scale a system.

One solution to optimize these space-filling mechanisms' drawbacks may be the usage of Voronoi diagrams. Since the Voronoi diagrams consist in irregular polygon forms, they can lead to a more efficient division of space, instead of the rectangular/quadrangular shapes used in geohashing, quad-tree hashing and CAN.

Voronoi nodes have knowledge of their indirect neighbors and of their neighbors' boundaries (does not happen in CAN and the others) which facilitates nearest neighbor discovery and makes spatial queries easier to perform (for instance, in [44], it is represented a solution that supports range queries working with Voronoi). In addition, this mechanism allows space's spherical representation, which cannot happen using the CAN (this feature will be discussed in a further section of this chapter). Figure 3.1 a) demonstrates GeoSharding using CAN partitioning and Figure 3.1 b) demonstrates GeoSharding using Voronoi spatial division, where the shards are represented as white squares. In CAN it is possible to observe a rectangular division of space where there several points inside a shard's region that are not assigned to its closest shard. In the Voronoi case, all the points inside the irregular polygons are assigned to its closest shard, the correspondent Voronoi polygon's generator.



a) GeoSharding using CAN

b) GeoSharding using VAST

Figure 3.1: GeoSharding using different spatial partitions

3.2 Voronoi Based GeoSharding

After studying the state of the art in NoSQL databases, in database partitioning and in geographic data storage mechanisms, the sharding algorithm proposed will be illustrated in this section.

The general architecture (represented in Figure 3.2) consists on modifying the object's sharding keys in a data storage system's Application Program Interface. The new keys would be based on the data georeferenced characteristics (the geographical location associated with the object). After, the data system's sharding, replication and querying mechanisms have also be implemented based in the new keys and in Voronoi Indexing, resulting in the GeoSharding solution pretended.

The GeoSharding solution achieved must respect the requisites already presented in existing systems. Data basic availability has to be assured, such as its consistency, which will be demonstrated along

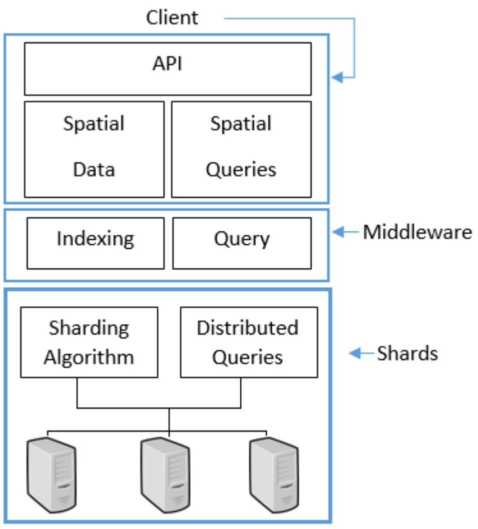


Figure 3.2: GeoSharding Mechanism Design [4]

this dissertation. The system must be partition tolerant, since it is based in a distributed environment, so it is difficult to achieve full data availability and consistency but the systems presented along the report reached balanced solutions between these three characteristics. In order to assure this goals, the Geosharding algorithm uses a proper replication mechanism that allows data availability and fault tolerance, that is discussed further in this chapter.

3.2.1 Shard's Assigned Positions

The shard key used to define data partitioning in Geosharding is based on the georeferenced positions of the objects to store in the database. The data partitioning among the shards depend on the spatial division/indexing mechanism used by the system, being each shard responsible for a unique region of space. To do so, it is necessary to assign a georeferenced positions to the shards. These positions consist on virtual coordinates that are chosen depending on the dimension of the geographical space involved, the load balancing taking into account data spreading across the space and the neighbors' positions (the other shards).

By assigning georeferenced positions to the shards, the benefits of GeoSharding improve database partitioning and this will be proved and evaluated in a further chapter of the work.

As it was stated before, a regular ruled partitioning of space like what happens in CAN or Quad-tree partitioning does not support well the option of assigning any desired position to the shards. Instead, the alternatives to implement these "picking" of positions to the servers would have to rely in choosing an interval of values for the pair of the coordinates so that the servers would end assigned to a position not very far from the truly desired position. The other option would be to sub-partition space multiple times, in rectangles and squares, so that it would be possible to have a shard in the desired position (but this would require more shards/servers than the ones available or desired).

This is an advantage of using Voronoi-based partitioning, since the irregular partitioning of space allows putting the Voronoi polygon centers and boders in any position desired. This comparison between Voronoi and the others for server assignment will be tested further in this dissertation.

3.3 Voronoi Based GeoSharding's Advantages

3.3.1 Hash Based Partitioning Vs Range Based Partitioning

In a distributed database environment, like the case of some NoSQL systems, there are two main groups of strategies that can be approached when implementing the System's Sharding mechanism. The choice between one or other is made taking into account the database characteristics that matter most to the client implementing it. These two approaches are called Hash Based Partitioning and Range Based partitioning.

For each of these strategies is necessary to choose the sharding key of the object, which can be an index field or an indexed compound field of the same object to store. The explanation given in this section about these two types of partitioning is made accordingly to the MongoDB's online manual [7], as this is one of the NoSQL databases allows choosing between one of these two mechanisms.

Range Based Partitioning consists in dividing "the data set into ranges determined by the shard key value to provide range based partitioning" [7], supporting more efficient range queries. Given a range query on the shard key, the query router can easily determine which shards overlap that range and route the query to only those shards. By querying only, a limited number of shards/servers is possible to optimize the range queries, reducing network communications.

However, Range Based Partitioning may cause unbalanced load distributions, as there are shards that may end up with much more data stored than others, depending on the uniform variation of values of the shard key among the data sets. This negates some benefits of sharding since in this situation, a small set of shards may receive the majority of requests and the "system would not scale very well" [7]. Figure 3.3 illustrates a scheme with four shards, each one assigned with a possible range of values assumed by the shard key.

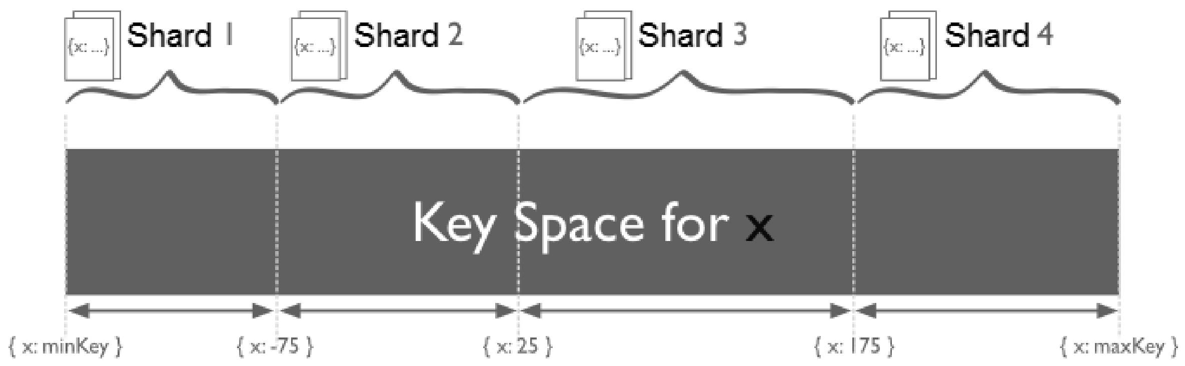


Figure 3.3: Range Based Partitioning Representation [7]

The other partitioning strategy is Hash Based Partitioning, that consists on hashing the shard key into a value and using this random value to locate the shard where to save the data. With this type of partitioning, two objects with approximate shard key's values are unlikely to be saved in the same shard. This fixes the load balancing problem because it guarantees a more random distribution, but range queries become inefficient as the system would more likely query every shard in order to return a result.

When the final system desired does not favours the range queries usage, this strategy (hash-based) is preferable as it guarantees load balancing among the shards which allows a better scalability of the system. Figure 3.4 represents a scheme with four shards, each one assigned with a possible range of values assumed by the hashed value of the shard key.

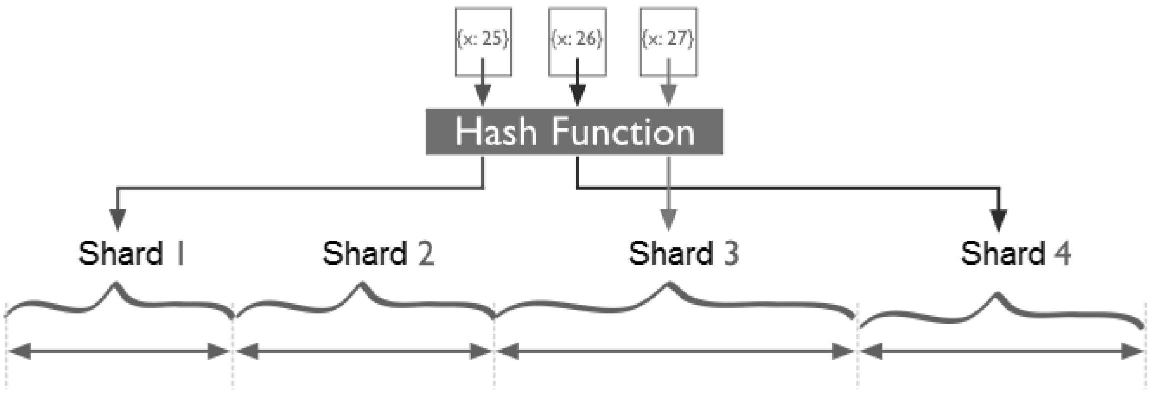


Figure 3.4: Hash Based Partitioning Representation [7]

However, by exploring the georeferenced characteristics of data, the GeoSharding policy proposed can solve this paradox by optimizing both range queries and load balancing. The spatial indexing mechanism used to implement GeoSharding influences both the characteristics that are desired to be optimized but this matter will not be revised yet again, being evaluated in a further section.

A good example of a spatial dataset is an Earthquakes dataset stored by the Northern California Earthquake Data Center [45]. The Earthquakes are a good example of georeferenced data spread irregularly across the globe (georeferenced objects with coordinates all around Earth, grouped in clusters), as exemplified in Figure 3.5. Now, it will be explained how GeoSharding could optimize the storage of the presented dataset. As it is possible to observe, the data points are spread irregularly across the space and grouped in clusters where there are more volumes of data.

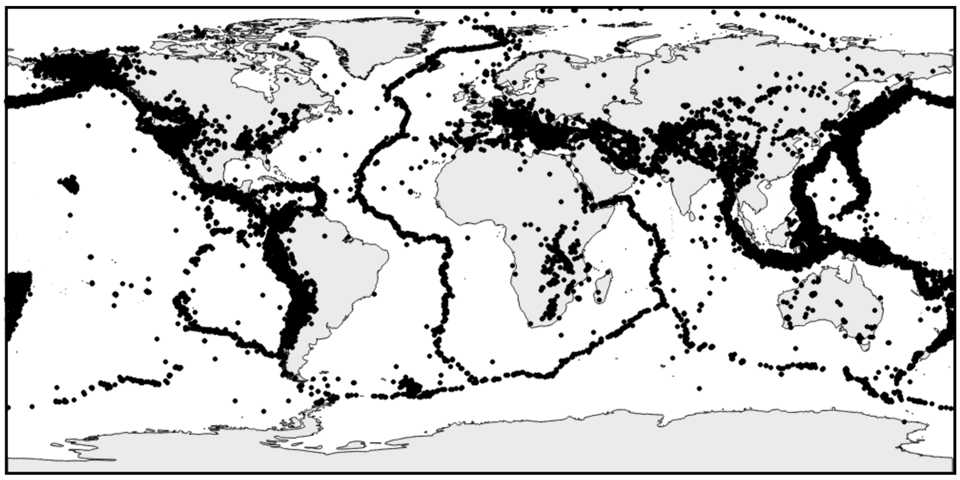


Figure 3.5: Global Earthquake Dataset

Since the GeoSharding mechanism proposed consists on attributing coordinates to each shard in order to optimize data partitioning among them, there must also be an optimal strategy to choose these same coordinates. The georeferenced data points are going to be stored in the server responsible for the spatial block the point is inside.

Figure 3.5 shows a global case, with data around the world, so it makes sense to attribute coordinates also irregularly spread around the world. With, for example, a server responsible for the data in each continent, the distribution would not be unbalanced and spatial range queries (typical query used in GIS systems) with a continental dimension (geographically speaking) would contact a limited number of shards to retrieve the data pretended. Figure 3.6 shows a possible distribution of 10 positions assigned to the shards across the globe.

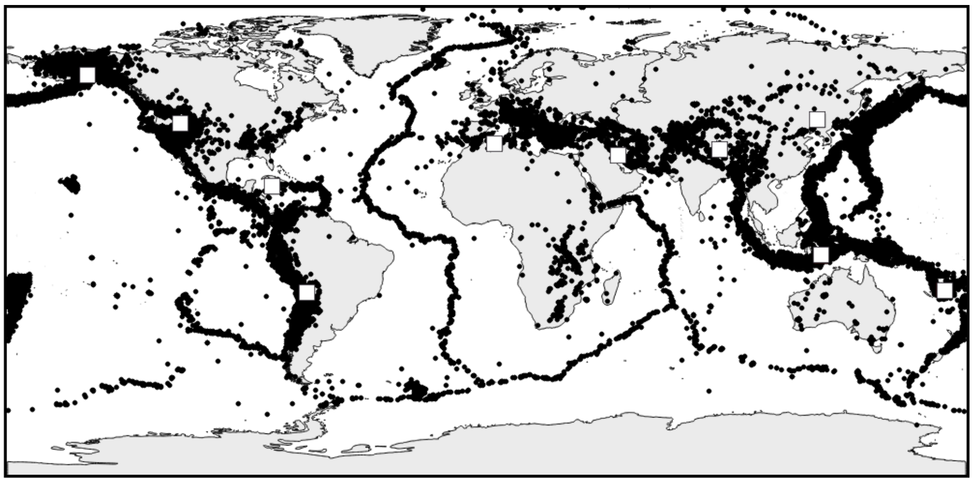


Figure 3.6: White Squares indicating the possible positions assigned to the GeoShards

This way, load balancing could be optimized without using random distribution of data (like in the Hashed Based Partitioning), and spatial range queries would also be optimized (which happens in Range Based Partitioning) which would solve the paradox to the databases which use georeferenced data. This optimization will be evaluated and proved in the next chapter.

3.3.2 Spherical VAST

Section 3.1 presented some of the disadvantages of other spatial partitioning techniques in comparison with Voronoi Diagrams. One of them, was that some of these data points that in reality were close to each other on Earth, would end up in the opposite side of the plane after the spatial partitioning.

This happens not only with Geohashing but also with other bi-dimensional spatial partitioning mechanisms like CAN and Quad-Tree Hashing. Other disadvantage of considering the bi-dimensional representation of Earth and working with georeferenced global data is that distance near the Poles is significantly different than near the Equator, which makes spatial assignment to the centers of the spatial division blocks very unbalanced and unrealistic.

In order to do so, the geographically distance function used by the system to define spatial division (regions inside each polygon are assigned to the Voronoi generators where the generator chosen is the nearest generator) has to guarantee that the Earth is treated like a sphere and not a bi-dimensional plane.

Since the coordinates assigned to the georeferenced objects are most of the time GPS coordinates (in this case decimal degrees of longitude and latitude), it would be necessary to calculate the real distance between two points in the Earth surface, defining a spherical Voronoi diagram (like represented in Figure 3.7).

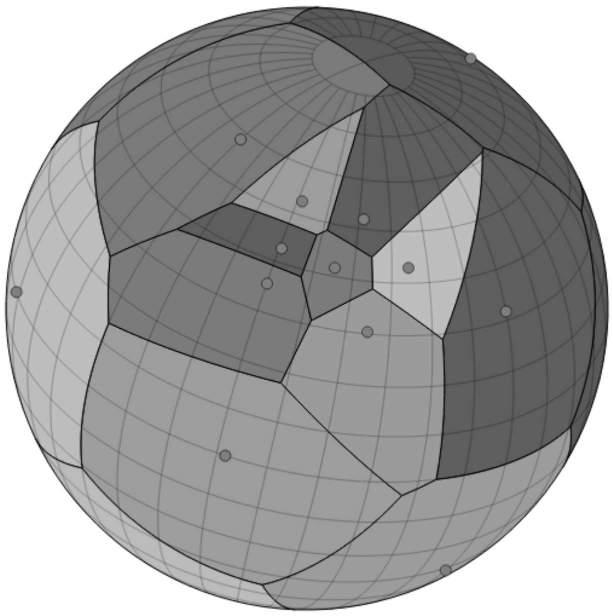


Figure 3.7: Spherical Voronoi Diagram

There is already a formula to calculate such distances, the Haversine Formula [46], that was implemented in order to calculate the great-circle distance between two points (which means the shortest distance over the Earth's surface).

Even though the Earth is not a true sphere, for this purpose, the solution presented is accurate enough as it solves most of the problems presented by the other bi-dimensional spatial indexing mechanisms, ignoring the ellipsoidal effects of the planet. The Haversine formula, given the latitude and the longitude coordinates, is given as follows:

$$a = \sin^2(\triangle\varphi/2) + \cos\varphi 1. \cos\varphi 2. \sin^2(\triangle\lambda/2)$$

$$c = 2. \arctan(\sqrt{(}a), \sqrt{(}1-a))$$

$$d = R.c$$

where φ is the latitude, λ is the longitude, R is the Earth's radius and d is the calculated distance between the two points. In order to compute the result correctly the coordinates' values have to be in radians.

Now, the bi-dimensional problem of the two points on opposite sides of the plane is solved, as the coordinates are used in a spherical way and the virtual distance between them is much more realistic than with the other spatial partitioning methods adopted to this case. Furthermore, the distances near the Earth Poles are not considered the same as the distances in the Equator. It is possible to observe such improvements in Figure 3.8.

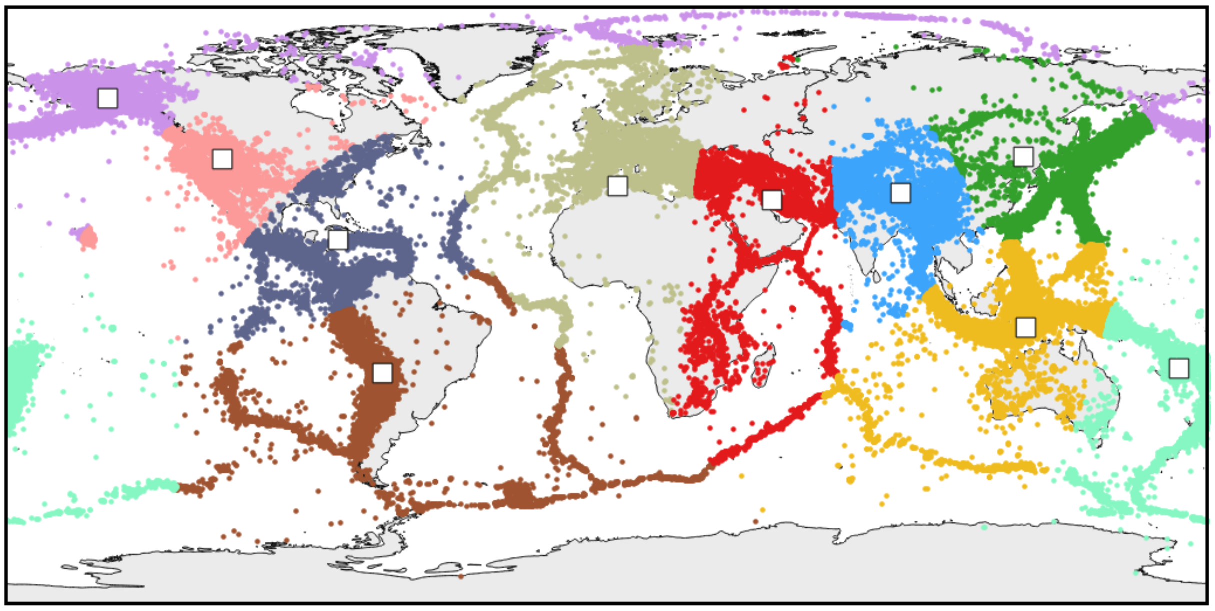


Figure 3.8: Dataset distributed by Spherical VAST

Figure 3.8 presents the servers' positions (white squares in the case of Figure 3.8) chosen and presented already on Figure 3.6 (white squares in the case of Figure 3.6), and the Earthquakes georeferenced data points (previously presented on 3.5) have been given a color to represent in which shard the objects are stored. This way is possible to show that VAST implemented successfully the spherical Voronoi divisions.

Of course this spherical version of VAST makes more sense to use in a global perspective. When dealing with data from one Country or one City, the bi-dimensional version is accurate enough. This can be a configuration parameter of the database using it.

3.3.3 Replication Mechanism

Section 2.2.4 discussed the importance of the Replication mechanism in NoSQL databases, where reliability and load balancing were two important motives that make this mechanism essential. Since there is no existing sharding mechanism based on the spatial coordinates of an object, it is also important to propose a suitable replication mechanism, that would also work with the spatial partitioning system used.

Assuming the Voronoi indexing scenario where each shard's assigned position on the diagram is a Voronoi polygon generator, the replication strategy proposed and evaluated consists in copying each data object to the n nearest Voronoi generators (shards), storing the object in other shards besides the nearest one, where n is the number of replicas desired in the system.

The goal of such replication system, besides guaranteeing reliability and fault tolerance (like in normal replication mechanisms implemented in actual NoSQL systems) is to optimize the number of shards requested to resolve a spatial query, since the objects are replicated taking into account its spatial position. This replication strategy may also work with the shard's position relocation (or when a shard of the system temporarily fails), since the objects are stored in the n "closest" shards.

This proposal will also be evaluated in a further chapter of this thesis.

3.3.4 Multi-Point Geometry Objects

Geometry Objects composed of multiple points or regions should also be taken into account. For instance, the route information of a given vehicle, such as city bus' routes or taxi routes, is composed a set of thousands of points.

If a distributed system and its sharding mechanism does not take into account the fact that such objects are composed by multiple coordinates, each object will be split by various shards. The access and processing of a complete object will lead to overhead. This overhead may be reduced if each shard responsible for some points of those geometry objects contains a complete copy of it.

In order to do so, we propose a second replication mechanism, specifically designed for multi-point geometry objects. In this replication policy, all the shards responsible for some point of a geometry object will contain a replica of the object.

By doing so, the insertion of this class of objects will incur some costs, but it is expected that spatial range queries will suffer no degradation and the queries that request a complete object will be optimized. This special replication mechanism is also evaluated further in the work.

3.4 Summary

This chapter introduced the proposed partitioning policy to optimize data partitioning in a spatial sharded database. GeoSharding uses the spatial object's coordinates as the index and sharding key to partition data between the shards, which results in each shard storing the spatial data inside its spatial region.

To solve the hash-based partitioning vs range-based partitioning paradox, in the case where these queries are spatial queries, GeoSharding uses Voronoi indexing which allows optimizing both load balancing and spatial queries' performance. In this way, Voronoi outmatches the other spatial indexing mechanisms regarding this sharding strategy's implementation.

A Replication mechanism based on Voronoi partitioning was also introduced, where the N replicas are stored in N closest shards (which geometrically are Voronoi generators). A special replication strategy for multi-point geometry objects was also introduced where the whole object is stored in all the shards intersected spatially.

In the next chapter we present the results that show that GeoSharding using Voronoi partitioning is a mechanism that optimizes database partitioning.

4 Evaluation

Contents

This chapter presents GeoSharding's simulation, the data sets used to test GeoSharding, the different algorithms evaluated, the test description of the experimental phase and the master thesis' results.

GeoSharding was implemented and tested in a simulated environment. The implementation in a real NoSQL database would be possible but alone would not prove database partitioning optimization. A simulator where more sharding algorithms were implemented and compared to the mechanism proposed, allows to show the partitioning optimizations.

Peersim [47] was used to simulate and implement this master thesis' work, a Peer-to-Peer network simulator capable of supporting addition and removal of nodes. One of the original goals of this simulator development was to test new peer-to-peer networks protocols, since most of the times, in the early stage of the development of a new protocol, it is not feasible to implement it on a real environment. It is a simulator capable of supporting a large number of network's nodes which is useful when it is not possible to test a new protocol with several nodes in a real environment.

4.1 PeerSim

PeerSim is written in Java and it supports two different simulation engines [47], a cycle-based and an event-driven engine. The cycle-based engine is simpler since it uses some simplifying assumptions: "lack of transport layer simulation and lack of concurrency". This means that the nodes communicate directly with each other and they are given the control periodically, in a sequential order (lack of concurrency) to perform arbitrary computations like calling methods of other objects. This engine supports better scalability but is more unrealistic.

On the other hand, the event-driven engine discards these assumptions, which lowers its efficiency but is more realistic. Both the engines are "supported by many simple, extendable and pluggable components", using a flexible configuration mechanism.

GeoSharding and the other algorithms evaluated were implemented using the cycle-based engine, since for the test purposes the network details implemented by the event-driven engine were not relevant and cycle-based engine is simpler. To develop all the mechanisms presented further in this work we also wrote all the code in Java.

PeerSim was designed to work with different building blocks (modular programming based on objects). To replace one block is simple, since it is only necessary to re-programme the same interface of the previous object, which has one determined functionality.

The general simulator life cycle consists on the following steps:

1. Choosing the network size (number of nodes in the network), which represent the number of shards that will be used to represent the different sharding mechanisms implemented;

2. Choosing one or more protocols to experiment with each node and initialize the protocols (correspondent to the different sharding mechanisms that will be used for testing);

3. Choosing one or more Control objects to monitor the desired properties and parameters of the network, which in this work corresponds to data locality and data access issues. These Control

points may also be used to modify network parameters, like the network size and the internal state of the network protocols;

4. Running the simulation by invoking the Simulator class with a configuration file as argument, that contains the information listed in the previous three steps.

During each simulation, all the objects that are created are instances of the main interfaces of PeerSim. These main interfaces are listed below [47]:

- Node: A node is a container of protocols and this interface represents each of the P2P network's nodes. This interface provides access to the protocols each node has, and a fixed ID of the node;
- CDProtocol: It is a protocol, which runs in the cycle-driven. Such a protocol simply defines an operation to be performed at each cycle;
- Linkable: Usually implemented by protocols, this interface specifies how each node has access to a set of neighboring nodes. An overlay network is defined by the instances of the same linkable protocol class over the nodes;
- Control: Classes which implement this interface can be scheduled for execution at certain points during the running simulation. These are the classes that typically observe or modify the simulation.

The first step in each simulation consists on reading the configuration file, given as a command-line parameter. This configuration file has all the information parameters concerning all the objects running on the simulation, and it is the only argument of the program. After, the simulation sets up the network by initializing its nodes, and each of the nodes' protocols. Each of these nodes have the same kind of protocols which means that instances of protocols form an array in the network (where there is an instance of each protocol for each node).

To initialize each protocol, it is necessary Control objects that are scheduled to run only at the beginning of each simulation. Once the initialization is over, the cycle driven engine calls all components (protocols and controls) once per cycle, until the number of cycles specified in the configuration file is over or until a component terminates the simulation.

In the algorithms implemented, each shard was a Node object containing a CDProtocol regarding each algorithm's different sharding strategy. The different algorithms evaluated are presented in the next chapter. In each simulation, during the initialization phase, each object of a chosen data set is stored in the Node's CDProtocol according to the sharding algorithm being simulated. This distribution of the objects of the data sets by the different Nodes is done taking advantage of the centralized control the simulator allows over the simulation's different Nodes. In each simulation were used 10 nodes in the network.

4.2 Data Sets Used

In order to test the Sharding strategy proposed, it was important to simulate it with real datasets stored by actual GIS systems. The datasets used during the simulations are described in this section.

The size of the geographical space that contains the georeferenced data stored by GIS' databases may belong to different categories. For example, a public transport network (bus, subway, ...) of a given city presents a local dimension and a social network's georeferenced data, which is spread geographically all over the world, presents a global dimension. Furthermore, the spatial data stored may be distributed in different ways across the space, for example, it can be uniformly distributed or irregularly distributed across the space.

Given that this thesis's goal is to prove that GeoSharding optimizes data partitioning for most of the types of georeferenced databases systems, the tests realized after implementing the sharding mechanism in a simulated environment used three different georeferenced datasets, in the scope of testing the policy with three different spatial distributions, all of them handled by its own responsible GIS system.

The radial distribution corresponds to a huge volume of data centered in one small region of space, and the volume of georeferenced points decreases with the distance from that center. The clustered distribution corresponds to several clusters of data spread across the space. The uniform distribution corresponds to a regular distribution of the georeferenced points across the space. The way the data points are spread across the space is much more relevant than the spatial scale of the datasets, since varying the scale with the same type of data points' distribution would originate the same results. However, the global scenario may be tested with a spherical representation, which represents an important difference.

In this evaluation, the data size of the datasets is irrelevant, but the number of spatial data points used during the simulations is relevant.

4.2.1 Local Data Set (Radial Distribution)

The first data set chosen had the goal to represent a local dimension (City georeferenced data) with a radial distribution (the volume of data is centered in some point of the space available, and it diminishes as the distance from that center grows). The sample data set used is a trajectory dataset that contains one-week trajectories of 10,357 taxis [48,49]. The total number of points in the dataset provided is about 15 million (we used a sample of 328000 data points) and the total distance of the trajectories reaches 9 million kilometers.

GPS-equipped taxis "can be regarded as mobile sensors" [48,49] probing traffic flows on road surfaces. This is a typical Data set of Big Georeferenced Data useful for providing route information as the taxi drivers are "usually experienced in finding the fastest route" to a destination based on their experience.

The presented taxi routes belong not only to the city of Beijing, in China, but also to the regions around the city. As it is possible to state, by observing Figure4.1, most of the data is located at the City Center, being this huge volume of data significantly bigger than the number of points spread across the region surrounding the city.

It presents an uneven radial distribution of georeferenced data across the map, whose access could be facilitated by an irregular partitioning of space. The data points of the data set are represented by a line which contains the taxi identifier, the date and the time and the longitude and latitude coordinates.

A second data set of Local Perspective was used only to test the geometry objects' replication mechanism proposed in section 3.3.4. This one is from the Dublin City Council's traffic control base [50] with

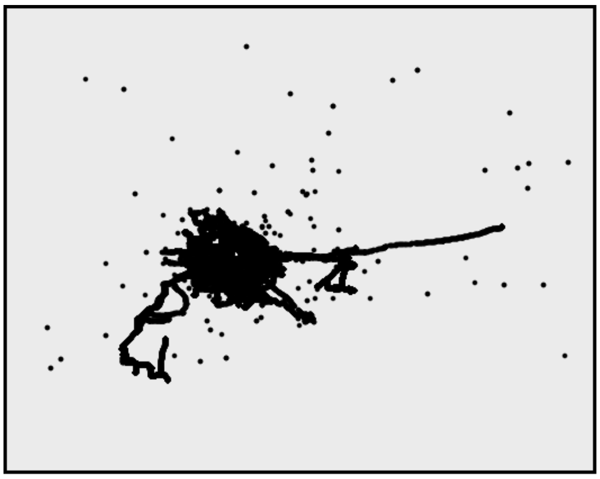


Figure 4.1: Beijing Taxi Routes' Data Set

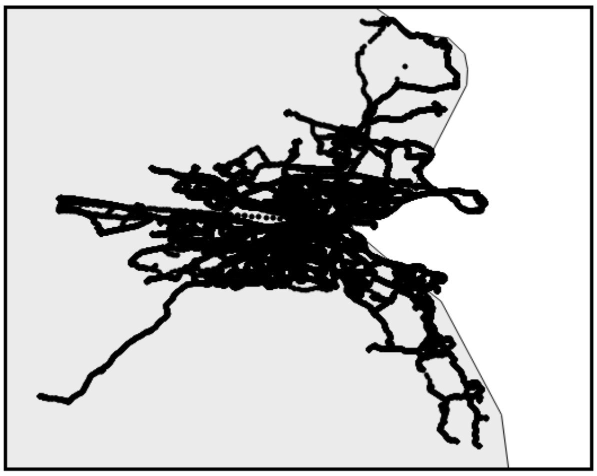


Figure 4.2: Dublin Buses Routes' Data Set

the routes of the city buses from a single day.

This data set is similar to the previous one but its advantage is that bus journeys, which are also composed of multiple data points, were identified by a unique ID, allowing the evaluation of the proposed replication for geometry objects. This allows to evaluate options to store multi-point geometry objects.

The data set is composed by 796000 data points, belonging to 5387 different journeys, and it is represented on Figure 4.2.

4.2.2 Continental Data Set (Uniform Distribution)

Several GIS systems handle data from a whole continent or a large country, so it was important to test the mechanism at this scale and with a new type of distribution: uniform distribution. As already pointed in the Spatial Big Data section, geological phenoms have occurrences that are stored in several GIS's databases for further study and analysis. As mentioned, some Government agencies like the United States Geological Survey [17], handle this type of data.

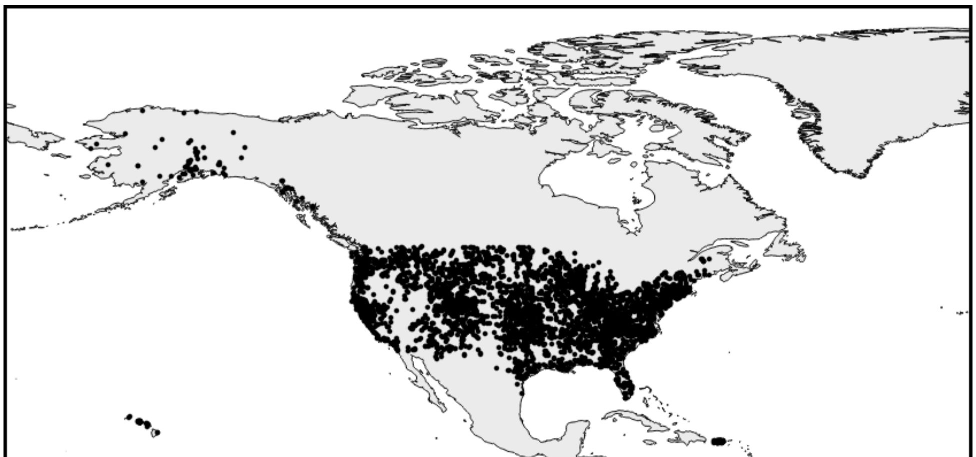


Figure 4.3: Underground Water Resources in North America Data Set

Between the data sets of earthquakes, volcanoes and landslides, that this agency stores [17], a data set of underground water resources which contains 4644 data points not only in continental United States of America but also in other regions of North America and in the Pacific Ocean was selected. Figure 4.3 presents this data set.

In this example, most of the data points are uniformly distributed across the USA mainland and very few information is presented in the other regions of the space.

4.2.3 Global Data Set (Clustered Distribution)

Already presented before, the Earthquakes data set (Figure 4.4) was chosen to study the mechanism at a global scale and with a clustered distribution. It is an Earthquake data set whose occurrences are from the years of 1878 to 1999 with 686000 georeferenced data points. The several clusters of data points are mainly spread in the West Coast of the American continent, the Pacific and Atlantic oceans and in the Mediterranean sea.

The data points presented in this CSV have data about the data and time of the earthquake, the latitude and longitude coordinates, its depth, its magnitude, the event ID, among other information.

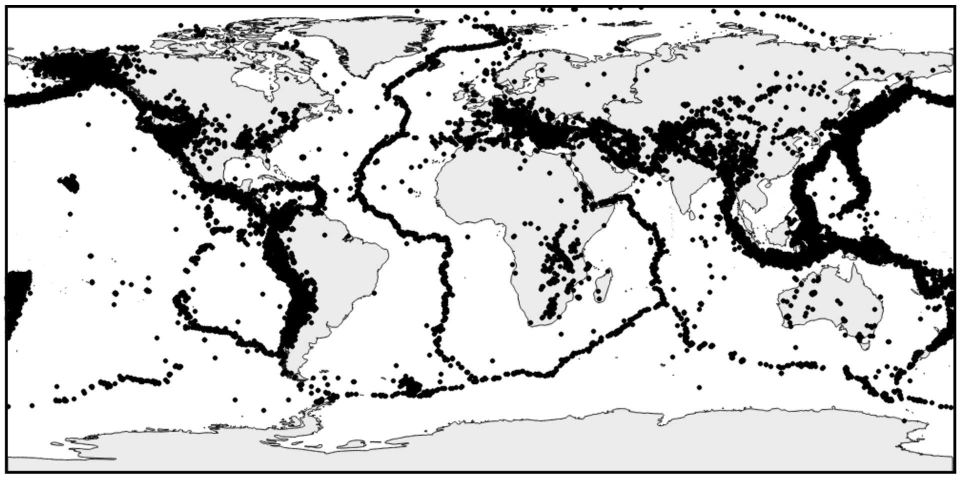


Figure 4.4: Global Earthquakes Data Set

4.3 Algorithms Tested

In order to prove that Geosharding optimizes data partitioning it is important to test it and compare it with actual Sharding mechanisms.

The design of this strategy was made to be implemented in a NoSQL database, in particular the ones that have a decentralized architecture, where all the nodes have the same responsibilities (there are no "master" nodes or "slave" nodes). Being so, the tests made also simulated a Sharding mechanism of a distributed environment used nowadays in order to compare its results with the policy proposed.

Section 3.3.1 referred that there are two main types of partitioning the data among the shards, each one having advantages and disadvantages. It was also stated that GeoSharding intends to optimize partitioning solving the paradox between these two types of partitioning (by exploring the georeferenced characteristics of data). So, since GeoSharding is more approximate to a new version of Range Based

Partitioning, it makes sense to test it against a Hash Based Partitioning and prove that it can outmatch it in the characteristics that usually only the Hashed Based Partitioning type has.

In order to represent a Hash Based Partitioning, the algorithm used was the Chord Algorithm [29], which is simple but represents most of the random distributions used in peer-to-peer networks. To guarantee load balancing, the actual NoSQL databases commonly use this type of partitioning.

4.3.1 Chord

The first tested algorithm in the experimental phase was based on Chord, whose description was presented in section 2.2.3.

To perform the distribution among the shards in the simulated network, each one of these nodes had a unique identifier (integer of 32 bits), generated randomly, and was responsible for the objects whose key value was in the interval between its own identifier and the previous neighbor's identifier (previous neighbor in the Chord ring).

For all the data sets used, to generate the object's key value in order to decide the shard where it would be stored, it was used the hash of the whole object (each data point can be represented as a String with several information). The hashing function used was MD5, to generate also an integer value with 31 bits (the object's key). This way, the object's distribution is totally random and is completely unrelated to its georeferenced characteristics, representing a version of the Hash Based Partitioning.

4.3.2 Chord+Geohashing

The second algorithm tested in the simulation phase was also a representation of the Hashed Based Partitioning, where the objects are randomly distributed among the existing Shards. It was used the same Chord algorithm described in the previous section, with the same mechanism to generate the shards in the network and its identifiers.

However, to reduce the randomness of the distribution and to start using the georeferenced information of each object, it was used a String containing the decimal degrees of the coordinates of latitude and longitude of each object to generate the hash which generates the key.

This way, the key values generated by the hash would be associated with the object's georeferenced characteristics and its randomness would be more limited. Note that this is not a version of the original GeoHashing, which uses spatial division and spatial indexing, but we called it Geohash since it is the hashing of the coordinates used. Along this dissertation this algorithm will be called "Chord + Geohashing". The goal of this change in the hashing calculus is to demonstrate the differences in the spacial queries and in load balancing that the georeferenced characteristics may cause, even in a Hashed Based Partitioning environment.

4.3.3 GeoSharding

The next two algorithms, presented and tested, implement the proposed partitioning solution called GeoSharding, which consist on using the geographical coordinates of the objects as the sharding key, storing the data in the servers "near" it.

For this purpose, both the algorithms use different spatial indexing mechanisms to partition space and work with the spacial characteristics of the objects to store. The choice of implementing it in more than one way allows the evaluation of the feasibility of a solution that works with Voronoi Indexing, a mechanism that is not used with actual NoSQL databases but which is the final solution proposed in this dissertation.

4.3.3.A GeoSharded CAN

In order to show the advantages of a Voronoi spatial indexing mechanism, we also implemented a simulated version of GeoSharding with a “regular” (rectangular/quadrangular) spatial division system.

The chosen partitioning mechanism was based on the Peer-to-Peer network named Content Addressable Network, already presented in the section 2.3.2. To implement CAN, it was used an open source version [51] of this distributed hash table that used as its base structure a Binary Spacial Partitioning Tree (BSP Tree), already presented in section 2.3.2.

Spatial partitioning using this type of tree results in a quadrangular and rectangular division of space, and the center of each square/rectangle is the position assigned to shard (the shard is responsible for storing the data whose coordinates are inside its square/rectangle). These structures are closely related to Quad-trees, presented in section 2.3.3, that partition space recursively into four sub-spaces.

As the number of shards in the simulation is limited, it is impossible to do unlimited sub-partitions with a low number if shards to “position” the shards where it is desired, which is a negative consequence of this “ruled” type of partitioning. It is possible to approximately position the shards in chosen zones of the space available, by dividing the space in a way that would allow it. Regarding the server distribution across the space, there will be a subsection dedicated to that step of the stimulation, further in this chapter.

4.3.3.B GeoSharded VAST

The last tested and implemented algorithm represents the final proposal in this thesis, the GeoSharding mechanism using Voronoi-based indexing. To implement this mechanism, the Voronoi Overlay Based Network, already presented, was used to test GeoSharding with the Local and the Continental Data Sets, and VAST with the spherical modifications (presented in section 3.3.2) was used to test GeoSharding with the Global Data Set.

Every shard in the system is treated like a Voronoi Generator Point and each one of them is responsible to store every data point inside its Voronoi Polygon. In order to determine the nearest generator point of the object, it was used the geometrical functions integrated in the open source version of VAST. To determine the positions assigned to the shards there were used three different methods, discussed in the next section.

4.4 Servers’ Positions Assignment

During the simulations with the different algorithms, we used 10 shards, as it is a number that can represent the number of data-centers (clusters of servers) of a georeferenced database. When discussing

GeoSharding, the positions assigned to the shards are as important as the spatial indexing mechanism of the system, since these two aspects together define which data each server stores. Being so, it made sense to test different ways of choosing the positions assigned to each shard. The three different distributions of shards across the space used were a random distribution, an uniform distribution and the hand picked distribution:

- The random distribution consists on attributing random coordinates to each shard in the simulation, being the shards distibuted randomly across the dataset space;
- The uniform shards' distribution consists on positioning each shard uniformly over the dataset, being each of them separated from the others with the same distance. In the CAN case, it represents the spatial partitioning used in the standard Geohashing, with a regular division of the space in rectangles with uniform measures;
- The Hand-Picked distribution consists on the strategy used to implement the final version of the GeoSharding proposed to optimize database partitioning. This way to assign positions to each server allows to "pick" the positions accordingly to what it is desired in terms of spacial queries performance and load balancing results (this will be discussed further in the thesis). Typically, the shards will be assigned to the spatial zones where there are huge volumes of data, which results in having no shard positioned in a zone where there a few or no data points to store, which would provoke an unbalanced network in terms of data load balancing.

Table 4.1: Shard's Positioning evaluation using CAN and VAST

CAN Shards	VAST Shards
Random Positioning	Random Positioning
Irregular Uniform Positioning	Uniform Positioning
-	Hand-Picked Positioning

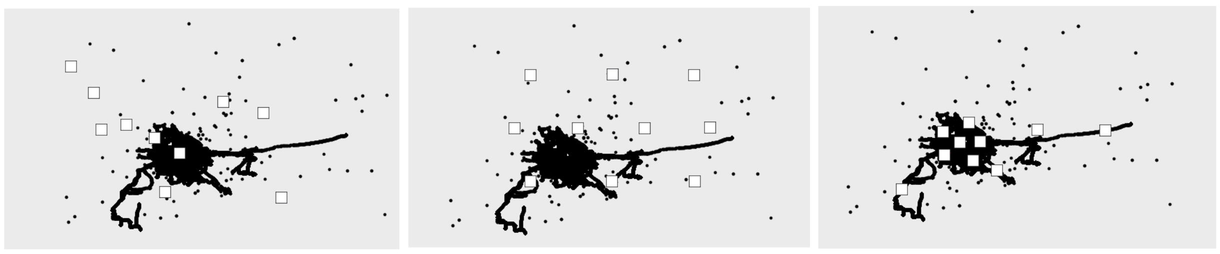

a) Random distribution b) Uniform distribution c) Hand-Picked distribution

Figure 4.5: Different Shard's Distributions

Table 4.1 resumes the different strategies tested regarding the positioning of the shards across the space, used by the two solutions implementing GeoSharding. Figures in 4.5 represent an example of the shards positioning strategies, regarding the Taxi Routes data set. Figure a) shows an example of a random assignment of positions to the 10 shards, Figure b) shows the uniform distribution of shards across the space and Figure c) shows the Hand Picked distribution chosen and used to partition the taxi routes data set.

Since CAN presents limitations due to its regular partitioning, the hand picked distribution was only used with Geosharded VAST and the uniform distribution was also only an approximation in the Geosharded CAN case.

4.5 Evaluation Tests

The goal of this mechanism of data partitioning is to optimize database partitioning, not only in terms of spatial data access but also in terms of load balancing in the distributed storage system used.

The two main strategies to partition data among the shards in the network, quoted from MongoDB's manual [7], were vital to understand the results of the tests performed. It is relevant to say that in the same Database's manual was referred that it is not possible to use a "geospatial index as the shard key index" which means that MongoDB does not implement a mechanism such as GeoSharding.

Both of these mechanisms have to compromise one of these characteristics: range based partitioning compromises load balancing and hash based partitioning compromises range query performance.

When discussing GIS systems, that store huge volumes of geo-data, is important to evaluate spatial queries' performance since spatial data access using this type of queries is very usual. Being GeoSharding a variant of Range Based Partitioning, since it uses a bi-dimensional range of values in the shard key, this type of query may also be considered like a bi-dimensional type of range query. To evaluate the performance of the four algorithms presented we used three types of tests.

4.5.1 Spatial Queries Performance

In order to test the spatial queries performance, the result regarding the number of shards requested in order to retrieve a spatial query result was the metric used to evaluate the four algorithms.

It was used a Bounding Box-type query, whose square region contain the data points that must be retrieved by the query. Since this spatial query may cover different areas of space, we made sure to vary the area of the Box intersecting the space, and also to select randomly the center of the square correspondent to the query.

To do so, the side of the square (measured in kilometers) was varied between short values (considering the space available) and values whose resultant product (box area) covered the whole geosharded space.

For each of these values (Query Square's Side) there were performed 100 queries, each of those with the box's center positioned randomly across the space, in order to have feasible results and to have a large sample set to calculate the results' charts. After performing the 100 queries it was calculated the average of the number of requested servers, and also its standard deviation.

Figure 4.6 presents an example of a Bounding Box Query, with the center of the query randomly generated and a fixed value of kilometers with the side of the Square, used with the Underground Resources data-set used.

The Spatial Queries test was performed using the 4 different algorithms presented, where the GeoSharded CAN was tested twice (with two different shard distributions across the space, random and "uniform")

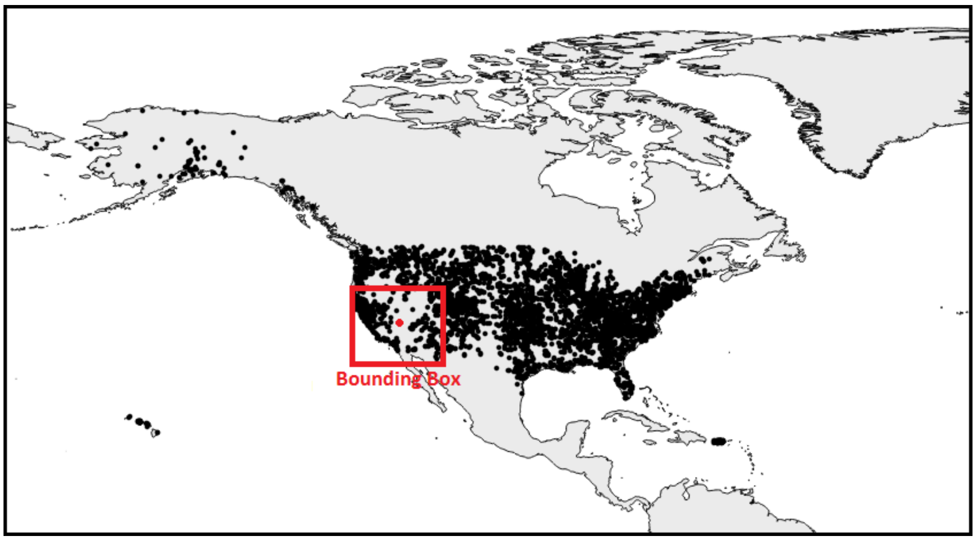


Figure 4.6: Example of Bounding Box Query, with a randomly generated center

and the GeoSharded VAST was tested three times (also with different configurations to spatially distribute shards, in random, uniform and "hand-picked" positions). This spatial query test mimics most of the geo-data processing algorithms that require the processing of adjacent data [22, 52].

4.5.2 Replication Tests

To test the standard replication mechanism proposed, only two algorithms were tested: Chord (representing the Hash Based Replication) and VAST (representing the GeoSharding's replication proposed). In the first case, each object was stored in the shard of the Chord ring responsible for the hash of the object and in the same shard's next neighbor in the Chord ring, simulating a random replication system where each object is stored twice in the database (the original and one copy). With VAST, each object was stored in the two nearest Voronoi generators, being one of them the shard whose Voronoi polygon contains the coordinates of the object stored (original owner of the object). Like in the previous tests, 100 spatial queries were performed, and for this matter was used the Earthquakes data set with VAST using its three different distributions of its shards.

To test the geometry object's replication mechanism, with the Dublin buses routes' data set, we used VAST with randomly distributed shards and queried the database to know what was the average number of shards storing each journey.

4.5.3 Load Balancing Performance

Since the Range Based Partitioning used in the Sharding mechanisms was known to optimize query performance and deteriorate load balancing, the second type of tests performed had the goal to evaluate the load distribution of data among ten shards in the network.

Being the first two presented algorithms, Chord and Chord+Geohahsing, a representation of the Hash Based Partitioning, it would be expected that these two strategies have a balanced load distribution among the nodes, so its results regarding this test were used as metric to evaluate the GeoSharding algorithms tested.

For each data set used, all the algorithms and its different shard distributions (regarding the GeoSharding mechanisms) were evaluated and the test consisted in computing the average of data that each shard would store in the network (being this average the same for all the implementations) and then compute the standard deviation of the data distribution among the shards.

4.6 Spatial Queries Performance Results

In the following sections, in each chart presented it will be discussed each algorithm regarding the Spatial Queries Performance and the Load Balancing Performance, in order to prove the optimization regarding GeoSharding.

First, it will be presented the queries' results, beginning with the random distribution of shards across the space available tests (correspondent to GeoSharding), followed by the results generated by uniformly distribute the shards across the space.

At last it will be evaluated the results regarding GeoSharding implemented by VAST "Hand-Picked" version, along with the maps correspondent to the Hand-Picked positions of the Shards chosen to optimize data distribution along the network. After, there will be a section regarding the Load Balancing performance of the tests also performed on the Simulator. In the end, there is a section regarding the Replication Mechanism's tests results.

To present the results correspondent to the spatial queries performance the student opted for a bi-dimensional chart where each point marked in the chart corresponds to the average (and the standard deviation) calculation of 100 spatial queries with each of the spatial query centers positioned randomly across the space.

The x-axis of the charts presents the Query's Box side measure variation, in kilometers and the y-axis of the charts presents the average number of requested servers/shards to retrieve the data requested by the spatial query.

4.6.1 Random Shards Distribution

Figure 4.7 a) presents the results correspondent to the Taxi Routes Dataset with the random distribution of shards (in the GeoSharding cases of CAN and VAST). As it is possible to observe, the tests regarding this data set had large values regarding each point's standard deviation on the charts, comparing to the other data sets' results presented further in this section.

When evaluating the spatial queries performance, the number of shards requested is significantly inferior in the Geosharded CAN and Geosharded VAST cases. In the same chart both of the lines of tendency corresponding to GeoSharding (CAN Random and VAST Random) are always below the lines of tendency of Chord and Chord+Geohashing, which means that clearly is necessary less shards/servers to retrieve the data requested by the bounding box intersecting the space.

Regarding the two methods representing the Hash Based Partitioning Type, Chord and Chord+Geohashing present slight differences in the queries with the Bounding Box's side measure at 24 km. This difference suggests that, even when randomly distributing the data among the shards, when the hash is only associated with the georeferenced coordinates the queries are slightly optimized.

When comparing CAN against VAST, CAN presents better results from the queries with square's side from 72 to 96 km, whereas VAST is only better in the queries with square's side at 24 km. This tendency will be compared with the other results presented. Regarding the randomly distributed shards

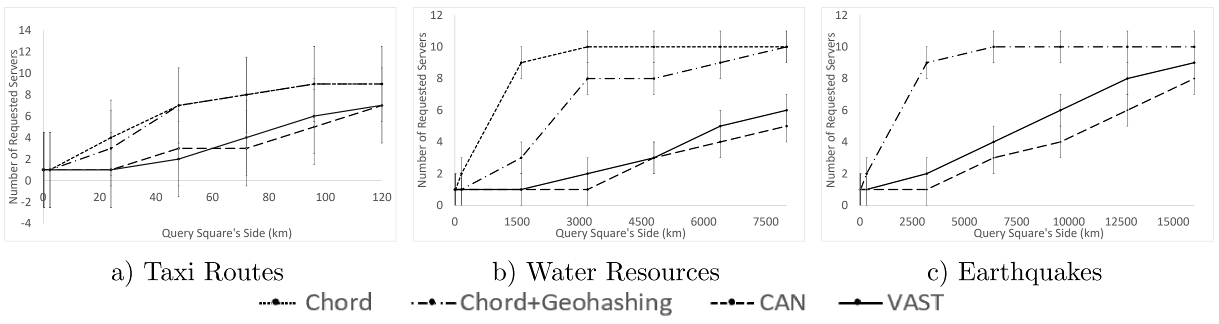


Figure 4.7: Spatial Query Results (Random Shards Distribution)

in the case of Underground Water Resources in North America, the chart in figure 4.7 b), there is a clear dominance in the Chord+Geohahsing case comparing it to Chord. The line of tendency of the first case is always below than the one belonging to the second, which indicates that the hashing associated with the coordinates of the objects gets a better random distribution than the hash of the whole object, even if it is still a random partitioning scenario.

The GeoSharding cases, once again, show better results than the Hash based Sharding cases, as the lines of tendency and the average points positions in the chart indicate.

The regular partitioning of CAN shows a slightly better performance against the irregular partitioning of VAST since there are more average points belonging to CAN that are below the points of VAST in the results chart.

The Earthquakes Data Set is the one which presents a more uniform distribution of data points across the space available and, due to this reason, the spatial queries performance results between Chord and Chord+GeoHashing do not present differences (Figure 4.7 c)). In the resultant chart there is only one line, correspondent to both cases, whose caption is indicated as Chord+Geohashing as the results are the same.

Spherical VAST looses against CAN in terms of performance, as its line of tendency is always above of the line correspondent to regular partitioning algorithm. The quadrangular queries do not benefit from the spherical boundaries of the Voronoi polygons, as it has more probability of intersecting more spherical and irregular polygons than quadrangular and regular divisions of space.

In general, the random distributed shards results presented, in the three data sets, similar results. The GeoSharding partition algorithms had always better performance than the Hash Based Partitioning cases, the Chord+Geohashing was slightly better than Chord and Geosharded CAN was slightly better than VAST.

4.6.2 Uniform Shards Distribution

In the following three charts, the GeoSharding cases present a uniform distribution across the available space. However, due to the "ruled" spatial partition of CAN, it was not possible to position the 10 shards exactly in a uniform way, as it would be necessary to have more shards to divide space in such a way that would allow it. Being so, they were distributed into different zones of space, trying to achieve an close to uniform distribution.

The Chord and Chord+Geohashing results are the same, so they will not be commented.

When evaluating the four algorithms with the taxi routes data set (Figure 4.8 a))the results between Chord and Chord+Geohashing and between Geosharded CAN and Geosharded VAST are very similar.

The first comparison was already analyzed, and the Geosharded CAN was only better than VAST in the queries whose bounding box's side measure was 48 km.

Once again the GeoSharding results present better spatial query performance than the Hash Based Partitioning, as it was expected.

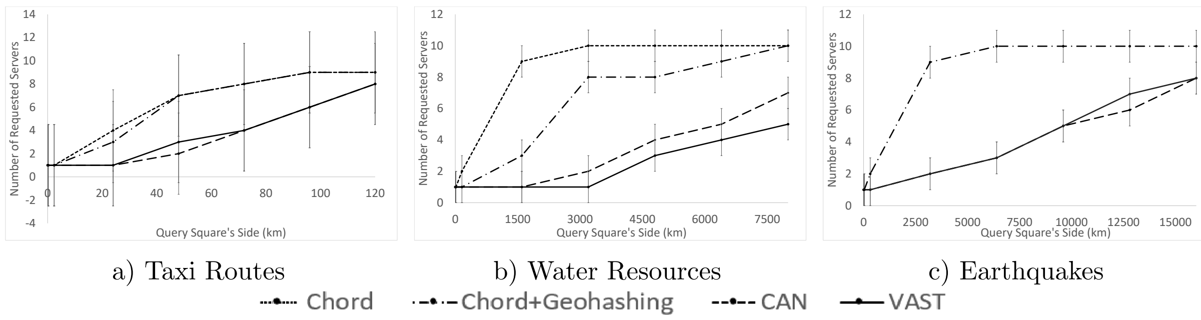


Figure 4.8: Spatial Query Results (Uniform Shards Distribution)

The Underground Water Resources with uniformly distributed shards (in the GeoSharding cases) presents a case where GeoSharded VAST had better spatial query performance than Geosharded CAN independently of the Query's Bounding Box Area, as the line of tendency of the first algorithm is always below the line of tendency of the second algorithm (Figure 4.8 b)).

Like the differences between CAN and VAST in the previous charts, this too is only a slight difference, which indicates that VAST may also be better than CAN regarding spatial query performance.

GeoSharding showed, here too, better performance as it needs much less shards to retrieve a requested spatial query's data.

The uniform distribution of shards in the Earthquakes set case presented partially the same query performance with VAST and CAN. Only in the case where the query box's side measure is 12800 km the average number of contacted shards of VAST (7) is superior to CAN (6) (Figure 4.8 c)).

For all the area values of the query's bounding box the GeoSharding cases showed a better performance than the Hash Based Partitioning strategies, which was already expected.

The general tendencies of the three charts presented with the results of the GeoSharding algorithms with uniformly distributed shards show that the sharding method proposed have always better query performance than the Hash Based partitioning methods, Chord+Geohahsing has slightly better performance than Chord and that CAN has slightly better performance than VAST.

4.6.3 Hand-Picked Shards Distribution

The following subsection contains the results correspondent to the performance of the final proposed solution to implement GeoSharding, with the positioning of the different Voronoi generators (shards' positions) chosen by the user in order to optimize data distribution across the shards. All the charts are presented alongside a figure of the data set map with the VAST shards' positions chosen by the student. The shards were positioned in the spatial areas with a great volume of data points.

As it is not possible to implement a hand-picked positioning solution in CAN due to its ruled partitioning mechanism, the final solution of the GeoSharded VAST was compared with the GeoSharded CAN with the randomly distributed shards' positioning. The other two algorithms (Chord and Chord+Geohashing) are also present in the charts, and, once again, the comparison between both will not be made as it was already presented in the previous subsections.

With the Taxis data set, the positions chosen for the geo shards of VAST were, in the majority of the cases, spread across the center of the city of Beijing. Since this central zone was the area with most of the data points of the data set, 7 shards were positioned in different central areas, in order to distribute the huge volume of data presented evenly among them. The other 3 shards were spread outside the city center to handle the data that was unevenly spread across the space available.

VAST had the same performance of CAN from the initial Query Box side's measures until the queries with square's side at 48 km (Figure 4.9 a)). Besides this point, Geosharded CAN presented a better performance. As GeoSharding mechanism, the Hand-Picked distribution of shards in VAST gave it a better performance than the Hash Based partitioning algorithms for all the ranges.

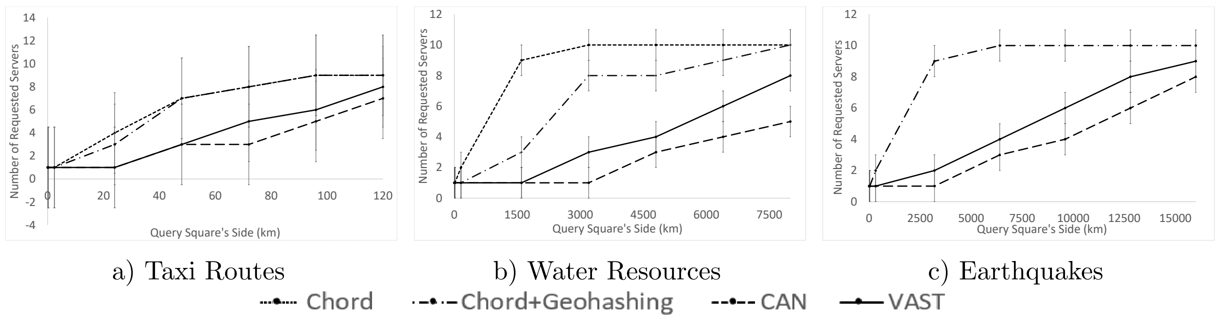


Figure 4.9: Spatial Query Results (Hand Picked Shards Distribution)

Most of the data points of the Underground Water Resources Data Set were spread across the United Sates country, so it was decided to uniformly distribute the 10 shards across the country as well. In this way, the huge volume of data was distributed evenly by each shard. The Costa Rica and Hawaii data points would belong to one of the shards located in the USA mainland.

The results (Figure 4.9 b)) show that VAST has the same performance of CAN until the point where the Query square's side has 3200 km. Besides this point CAN is better than VAST Hand-Picked.

As a Geosharded solution, VAST presents, once again, better results regarding spatial queries when compared to Chord and Chord+Geohahshing.

Regarding this global set, and as it was already presented before, the student positioned each of the 10 shards in different continents and regions of the globe, in the zones where it is possible to observe higher volumes of data points (Figure 4.9 c)).

Once again, when using Spherical VAST, quadrangular queries intersect more irregular and spherical Voronoi polygons than the rectangular forms of CAN, which makes the second case the optimum scenario regarding the spatial queries.

In all the 9 charts presented, it was registered a general tendency between the 4 algorithms, independently from the different data sets tested or the different shards positioning across the space:

1. The GeoSharding algorithms showed a better spatial query performance independently of the area of the query bounding box;
2. The resultant hash from the object's coordinates (Chord+Geohashing) had slightly better results than Chord in the continental and local data sets;
3. Geosharded CAN presented slightly better performance than Geosharded VAST in some cases, which can be explained by the irregular form of the polygons dividing the available space that cause more intersections with the regular square of the intersecting query. However, VAST will present a significant difference in the Load Balancing results that will outmatch this slight difference in the Spatial Queries performance.

Furthermore, to prove that the **Spherical VAST** (Global Data Set) presents a significant advantage regarding spatial queries when comparing it with CAN, a different new spatial query test was performed. By performing spatial square queries centered in the North Pole region with the bounding box's side measuring 4442 km, the results showed that, in average, CAN requested results from 9 shards (almost all the network shards) and VAST requested results from 5 shards (half of the network's shards). This proves the advantage of treating the Earth like a sphere when processing global spatial data.

4.7 Load Balancing Performance

The following section presents the three charts correspondent to the Load Balancing results obtained from the simulations performed and already described before, where each data set was stored using the different algorithms with spatially different shards' distributions. In each chart there is a result with the standard deviation of 7 different algorithms, which are: Chord, Chord+Geohashing, CAN with shards randomly distributed, CAN with shards "uniformly" distributed, VAST with shards randomly distributed, VAST with shards uniformly distributed and VAST with shards' positions hand-picked. In each shard it is also presented the average value of data points stored in each shard.

4.7.1 Local (Radial) Data Set

Figure 4.10 a) show the chart with the results regarding the Taxi Routes data set. To evaluate the feasibility of the GeoSharding algorithms, they will be compared to the Chord's results and the Chord+Geohashing's results, since these two are the reference regarding an acceptable standard deviation of objects stored in each shard. It is assumed that the random distribution resulting from hashing the whole object's string or just its coordinates gives the sharded network a balanced load distribution, since this is the advantage of using a Hash Based Partitioning Policy.

When analyzing CAN with randomly distributed shards it is possible to say that it has an acceptable load balancing result, since the standard deviation value is between the values of the Chord methods. However, CAN with uniformly distributed shards shows the worst performance between the 7 hypothesis, which clearly indicates an unbalanced network.

Regarding its random and uniform shard distribution, VAST presented satisfactory results since its standard deviation values were similar to the Chord methods. This means that the spatial query performance optimization does not compromise load balancing in the network.

On the other hand, the final solution proposed presented a significant optimization regarding all the other six algorithms proposed. The standard deviation of objects stored using the Hand-Picked shard distribution with VAST has a value similar to the average number of objects stored in each node. This value is clearly below the standard deviation values of the Chord mechanisms which indicates a significant optimization in load balancing, that was the only advantage of using random distribution of data points among shards.

This was expected since the shards were positioned strategically not only to optimize spatial queries but specially to optimize load balancing in the network. In this case, the majority of the shards were positioned in the central area of the city of Beijing (the area where there was a huge volume of data points) and divided among them the data in a balanced manner.

4.7.2 Continental (Uniform) Data Set

Figure 4.10 b) shows that a random distributions of the shards with both cases (CAN and VAST) cause an unbalanced network in terms of data load. Both the standard deviations for these distributions have a higher value than the standard deviations of the Chord methods.

Between both Chord methods, Chord+Geohashing presented a better result, although the difference is not significant, since the values of the standard deviation are very similar.

When observing the standard deviation of CAN with the irregular uniform distribution, it can be stated that it presents a balanced network, since its standard deviation value is very similar to the Chord+Geohashing method. However, VAST with the uniform distribution beats all the previous mentioned algorithms, presenting an optimization in Load Balancing since the values are already very satisfactory (close to the average of objects stored in each shard).

Finally, the best result belongs to the proposed solution to implement GeoSharding, that is VAST with the shards positions exactly chosen to optimize the load distribution along the network. Among the three charts of the load balancing results, this one shows the best case scenario regarding the "Hand-Pick" distribution, where its value is significantly below the average of objects stored in each shard. This means that the shards distributed across the country of the United States of America give VAST an important optimization in load balancing, being this load distribution much more balanced than with the Hash Based Partition algorithms.

4.7.3 Global (Clustered) Data Set

With the Earthquakes data set case (Figure 4.10 c)), all the other six algorithms presented a better performance than Chord, whose randomness resulted in a not very balanced network. Since Chord+Geohashing presented a better load balance regarding Chord, it is used in the section as the main Hash Based Partitioning algorithm to evaluate the other 5 Sharding solutions.

CAN with a random distribution was unbalanced compared with Chord+Geohashing, but CAN with the "uniformly" distribution presented a balanced network (similar results to Chord+Geohashing). However, all the three VAST solutions presented a better load balancing than the CAN and the Chord solutions.

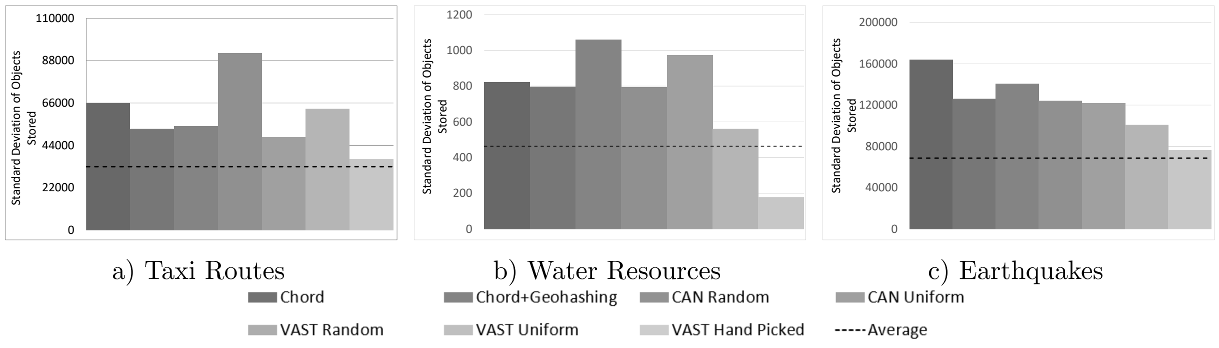


Figure 4.10: Load Balancing Results

Once again, the irregular spatial partitioning of VAST shows that it optimizes the load balancing in GeoSharding, being the hand-picked positions assigned to the shards the optimal solution to optimize this database requirement. In all the 3 charts presented in this section was registed a general tendency:

1. The Chord and the Chord+Geohashing algorithms, used as example to evaluate the load balancing across the network in the GeoSharding solutions, presented very similar results (except in the Earthquakes Data Set case, where the hash of the object's coordinates presented a better load balancing in the network);

2. The Hash Based Partitioning solutions, in general, presented a network better balanced than some of the CAN solutions, which means that CAN implementing GeoSharding does not improve at all the load balancing along the network.

3. The VAST solutions presented optimizations regarding the CAN and Chord solutions, being the Hand-Picked VAST (the proposed solution to implement GeoSharding) the one where this optimization is very clear and significant.

This section proved why VAST is the only spatial indexing mechanism that can implement an optimal version of GeoSharding, bringing optimization to both spatial queries and load balancing and solving the paradox of Range Based Partitioning vs Hash Based Partitioning in databases that handle big volumes spatial data.

4.8 Replication Results

Figure 4.11 present the result from the tests performed using the standard replication mechanism proposed.

Regarding the results presented, in the three different shards' distributions, using the Earthquakes data set, the spatial queries performance was optimized (when using VAST) by the new spatial replication strategy, being this fact clearly evident in the larger queries (query square's side measuring high values) where the number of shards requested to resolve the queries is lower in the replicated cases than in the cases where it is used VAST without replication. The random replication implemented using Chord did not improve the spatial query performance at all. This proves that replication based on geographical characteristics of data can improve spatial data access performance.

Regarding the geometry object's replication tests, each bus journey was replicated in average by 2 shards. The minimum number of shards storing a journey was 1 and the maximum number of shards storing the same journey was 5. This shows that is possible to replicate the data points of a route database in a efficient way to allow only one shard to retrieve a whole journey and at the same time optimize the spatial query performance tested before with different data sets.

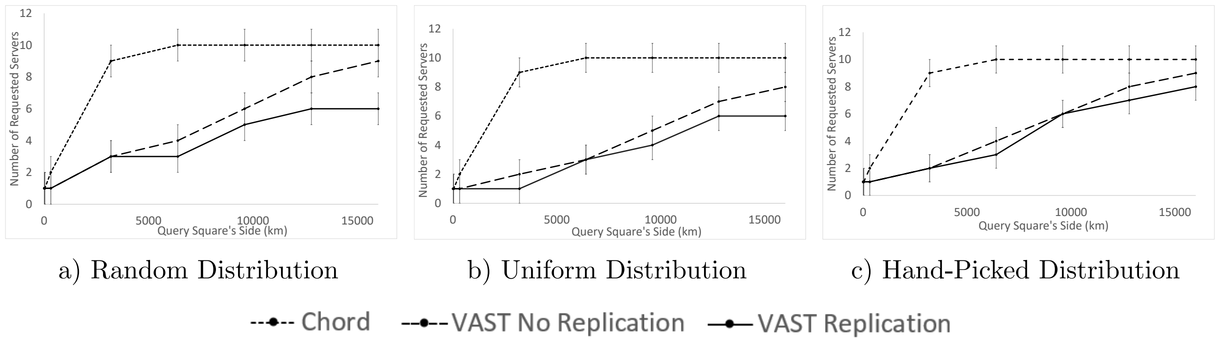


Figure 4.11: Replication Spatial Query Results (Earthquakes Data Set)

4.9 Discussion

After presenting the experimental evaluation's results of the mechanisms proposed, it is important to discuss certain aspects regarding GeoSharding, its implementation and its performance regarding the different algorithms tested.

It was possible to observe that GeoSharding did not compromise load balancing in order to achieve spatial query's optimization, which represents better data availability and less network communication traffic since less servers are required to resolve a query. In fact, the load balancing results when using the Voronoi polygons with the hand-picked shard positions showed also a slight optimization regarding the standard algorithms used in the NoSQL systems, which use Hash Based Partitioning to guarantee a balanced network.

It is also important to discuss why a Voronoi solution is optimal to implement GeoSharding. One of the most significant differences between the GeoSharded CAN and the GeoSharded VAST resided in the shards' positioning, because in the second one it is possible to choose exactly the positions assigned to the shards as there is an irregular spatial partitioning.

Regarding the GeoSharded CAN, as it was not possible to partition the space in the way desired since there was a limited number of shards (10), we used two different ways to assign positions to the CAN shards: a random distribution and a uniform distribution.

In the uniform scenario, it was only possible to try to partition space positioning the shards in spatial regions that were not too close to each other but it was not possible to do this with every shard. The random scenario was simple, since it consisted in generating random positions until it was possible to insert the node in the CAN BSP Tree, assigning the accepted position to a shard.

Regarding VAST, the random positioning assignment consisted in using the random positions assigned to CAN, to have a fair evaluation of both partitioning mechanisms, and assign them to the Voronoi Generator Shards. There were two more strategies to choose the server's positions: uniform distribution and the hand-picked distribution. Unlike CAN, the uniform positioning of shards in VAST was truly uniform since it was possible to exactly position the shards uniformly across the space available.

However, the third positioning evaluated with VAST (the Hand Picked Distribution) was the one presenting the best results regarding an overall evaluation between spatial queries' performance and load balancing. Since CAN had the limitations already mentioned, this hand-picked positioning was only possible in the Voronoi-based solution.

This way the Voronoi based solution was the only one that allowed to choose the positions attributed to the shards and that guarantees the best results regarding network's load balancing, guaranteeing also good results regarding the spatial queries' performance.

5 Conclusions

Contents

In this dissertation it was introduced an approach to optimize the database partitioning mechanisms of systems that work with Georeferenced Big Data. GeoSharding was defined and tested in a simulated environment, and its implementation on top of Voronoi was the only solution to optimize not only the spatial range queries but also the load balancing in a sharded system, solving the paradox of Hashed Based Partitioning vs Range Based Partitioning when working with georeferenced datasets.

This way, in GIS systems, it is possible to optimize data access and data storage by exploring the georeferenced locality characteristics of data. The sharding strategies used by actual sharded systems that manage spatial data are not adequate since they do not take any advantage of the georeferenced characteristics. GeoSharding is particularly useful in these scenarios.

Different options available to implement GeoSharding were evaluated, where the ones using common existing spatial mechanisms (CAN, Quad-tree) presented inefficiencies. By using a Voronoi-based solution, it was proved that spatial indexing mechanisms can be further optimized, specially when load balance and multiple sub partitioning avoidance are a priority. Using Voronoi diagrams, it is possible to avoid the fixed sub-partitioning of space and adapt the spatial network to the irregularities of the georeferenced datasets, by positioning each shard exactly where it is desired (allowing load balancing optimization).

VAST also allows its spherical implementation using the Haversine formula that is very useful regarding global georeferenced data sets, since it implements a spatial index that treats Earth like a sphere and not like a a bi-dimensional plane. This guarantees continuous data locality, even in the earth poles and around the 180^{th} Meridian.

It was also proved that it is possible to implement a replication system based on Voronoi diagrams that is simple, efficient and that can also optimize spatial data access (when using spatial queries). A special replication mechanism to handle Geometry Objects was also introduced and it was proved that a system can shard its spatial data to allow spatial queries and load balancing performance with GeoSharding, and at the same time optimize queries that request a whole Geomtery Object (only one shard requested in these cases).

5.1 VAST vs Virtual Servers

Along the dissertation, it was stated that CAN required multiple sub-partitioning to position shards in the exact desired locations, to allow better load balancing in the sharded network. In fact, this could be done by assigning each one of the multiple sub-partitions of space to virtual shards, that would be controlled by the actual network's shards. In order to perform this, it would be necessary a centralized control over the virtual shards to know which shard would store the virtual shards, which brings more complexity to the system.

However, by performing this multiple sub-partitions to optimize load balancing, the spatial queries' performance would be significantly worse, since a higher number of virtual shards and normal shards would have to be contacted to resolve the queries.

Once again, the solution achieved with VAST presents less complexity in the shards' position assignment and better results by optimizing both load balancing and spatial queries' performance.

5.2 Future work

The GeoSharding algorithm's feasibility was proved in this dissertation using a simulation software. In the future this sharding strategy can be implemented in a real NoSQL database that works in a distributed environment.

It is important to discuss the centralized database implementation vs decentralized database implementation, since there are NoSQL databases with these two types of architecture. The work developed along this master thesis considered a scenario where the shards/servers in the network did not have different roles while handling the georeferenced data to store which is a characteristic of the decentralized database architectures (GeoSharding can work with this kind of architecture).

However, the centralized control that PeerSim has over the nodes in the network during the simulation showed that it can also exist master nodes that manage the slave's nodes information, regarding its position in the space and the stored amount of data. In fact, a centralized control of the distributed database may privilege the dynamic load balancing already refereed, which is a positive aspect that shows that GeoSharding may also be implemented in a decentralized database system.

5.2.1 Dynamic Load Balancing

GIS system' databases are updated in a daily basis and by analyzing the data already stored it is possible, most of the times, to predict where the huge volumes of data will be positioned in space. In this case, it is useful, in the scope of GeoSharding, to assign positions to the servers accordingly to the spatial zones where most of the georeferenced data will be positioned.

However, as these systems store new data every day, sometimes there are unpredictable changes in where new data points to store will be generated. In these cases, an optimal solution found to position the shards in the virtual space may no longer be optimal anymore as the system may become unbalanced with the massive creation of georeferenced data to store in new spatial zones that once were semi-empty.

Furthermore, even if the shards were spread accordingly to the spatial zones where there were huge volumes of data, sometimes one of these zone's data density may significantly grow more than the others, causing a load unbalanced system. Once again, this suggests that an optimal solution that was found before to position the shards may not serve its purposes continually.

The ideal case, considering the suggested sharding policy, is that the shards would move/change its positions according to what is happening with the data to store in the system. This way, an overloaded shard could share its data as its neighbors' positions would approach its own position.

This idea could be explored in the VAST environment, as data partitioning in this spatial network is dynamic and not static (it does not imply multiple sub partitioning to choose the exact position of the center of each block, like in the other spatial indexing mechanisms).

Regarding this server mobility hypothesis, VAST was developed taking into account Gaming Environments, as explained in section 2.3.4. It supports the moving of every Voronoi generator in the network since it has a system of neighbor's updating (a node sends updates to all its AOI neighbors when it moves).

Even supporting this possible mobility of shards, the data transfer between servers would have to be developed considering the main requirements of a NoSQL Database System, like data availability and partitioning tolerance (two of three characteristics of the CAP theorem). It could also be used when a server or a shard becomes unavailable in the network (node failure).

This would not be the first time that a Voronoi-based approach would be used with objects' mobility. As it was already presented in section 2.3.4, ToSS-it is a spatial indexing mechanism for moving objects that support queries efficiently and also cope with frequent updates. Once again, Voronoi spatial division is preferred instead of a CAN approach.

If the Hand-Picked solution presented the best results among the tested algorithms, then, a developed algorithm that would dynamically position the shards in VAST to optimize the load balancing results would position the shards with a similar strategy to the ones hand picked.

5.2.2 Hierarchical VAST

Using GeoSharding with Voronoi partitioning to perform spatial division and attributing each spatial division to a cluster of shards is other hypothesis that could be tested in the future. Inside each cluster of shards, where the spatial division would be less relevant, the data partitioning could be handled using a Chord based algorithm.

This way, the algorithm would consist in routing to the shard clusters desired to resolve the spatial queries using Voronoi partitioning. After, the right shard in the cluster would be contacted using the classical hash-based partitioning, assuming a different communication cost. This hypothesis is illustrated by figure 5.1.

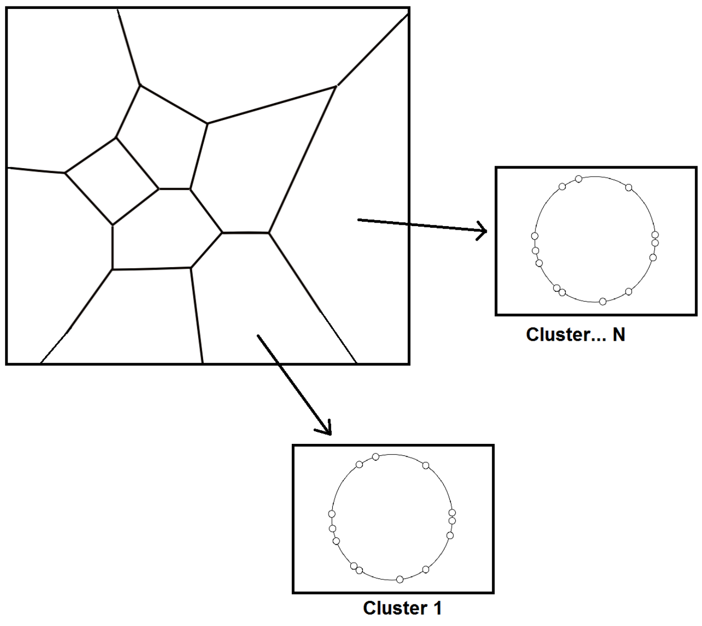


Figure 5.1: Hierarchical VAST with Sharded Clusters

References

[1] Abhishek Sagar and Umesh Bellur. Distributed computation on spatial data on hadoop cluster - technical report. *Dept. of Computer Science and Engineering Indian Institute of Technology, Bombay*, 2011.

[2] Egemen Tanin, Aaron Harwood, and Hanan Samet. Using a distributed quadtree index in peer-to-peer networks. In *The VLDB Journal*. Springer, 2007.

[3] VAST Development Team. Voronoi-based overlay network (von). www.vast.sourceforge.net. Accessed: 2016-6-17.

[4] Robbie Strickland. *Cassandra High Availability*. Packt Publishing Ltd, 2014.

[5] Project Voldemort. Project voldemort a distributed database. www.project-voldemort.com. Accessed: 2016-6-17.

[6] Carol McDonald. An in-depth look at the hbase architecture. www.mapr.com/blog/in-depth-look-hbase-architecture. Accessed: 2016-6-17.

[7] MongoDB. Sharding introduction. www.docs.mongodb.com. Accessed: 2016-6-17.

[8] Eric Brewer. Towards robust distributed systems. In *Conf.: Proc. of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*. ACM, 2000.

[9] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*. ACM, 2007.

[10] Raj Singh. The nosql geospatial landscape. IBM Cloud Data Services, www.slideshare.net/rajrsingh/geo-no-sql-42484582. Accessed: 2016-9-1.

[11] Guofeng Cao, Shaowen Wang, Myunghwa Hwang, Anand Padmanabhan, Zhenhua Zhang, and Kiumars Soltani. A scalable framework for spatiotemporal analysis of location-based social media data. In *Computers, Environment and Urban Systems*, volume 51, pages 70–82. Elsevier, 2015.

[12] Thomas M Connolly and Carolyn E Begg. *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.

[13] Pouria Amirian, Adam C Winstanley, and Anahid Basiri. Nosql storage and management of geospatial data with emphasis on serving geospatial data using standard geospatial web services. 2013.

[14] Xiaofang Zhou Yu Zheng. *Computing with Spatial Trajectories*. Springer, 2011.

[15] Inc OGC. Open geospatial consortium. www.opengeospatial.org. Accessed: 2016-6-17.

[16] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. Geospark: A cluster computing framework for processing large-scale spatial data. In *SIGSPATIAL '15 - Proc. of the 23rd Int. Conf. on Advances in Geographic Information Systems*. ACM, 2015.

[17] U.S. Geological Survey. U.S. geological survey. www.usgs.gov. Accessed: 2016-6-27.

[18] British Geological Survey. British geological survey. www.bgs.ac.uk. Accessed: 2016-6-27.

[19] Jie Bao, Yu Zheng, and Mohamed F. Mokbel. Location-based and preference-aware recommendation using sparse geo-social networking data. In *SIGSPATIAL '12 - Proc. of the 20th Int. Conf. on Advances in Geographic Information Systems*. ACM, 2012.

[20] Ciprian Dobre Nik Bessis. *Big Data and Internet of Things: A Roadmap for Smart Environments*. Springer, 2014.

[21] Daniel W Goldberg. A geocoding best practices guide. https://www.naaccr.org/, 2008. Accessed: 2016-6-17.

[22] Tomislav Hengl, Gerard B. M. Heuvelink, and David G. Rossiter. About regression-kriging: From equations to case studies. Pergamon Press, Inc., October 2007.

[23] Christof Strauch, Ultra-Large Scale Sites, and Walter Kriha. Nosql databases. *Lecture Notes, Stuttgart Media University*, 2011.

[24] Brad Fitzpatrick et al. Memcached. www. memcached. org, 2013. Accessed: 2016-6-17.

[25] Ricky Ho. Nosql patterns. http://horicky.blogspot.pt/2009/11/nosql-patterns.html, 2009. Accessed: 2016-6-17.

[26] Todd Lipcon. Design patterns for distributed non-relational databases. http://www.123seminarsonly.com/Seminar-Reports/2013-02/106435788-62816089-No-SQL.pdf, 2009. Accessed: 2016-6-17.

[27] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proc. of the twenty-ninth annual ACM symposium on Theory of computing*. ACM, 1997.

[28] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*. ACM, 2003.

[29] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM Computer Communication Review*, volume 31. ACM, 2001.

[30] Mike Malone. Scaling gis data in nonrelational data stores. http://www.slideshare.net/mmalone/scaling-gis-data-in-nonrelational-data-stores, 2010. Accessed: 2016-6-17.

[31] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Procs. of the 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 161–172. ACM, 2001.

[32] Anirban Mondal, Yi Lifu, and Masaru Kitsuregawa. P2pr-tree: An r-tree-based spatial index for peer-to-peer environments. In *EDBT Workshops*. Springer, 2004.

[33] Bruce Naylor et all. Bsp tree frequently asked questions. ftp.sgi.com/other/bspfaq/faq/bspfaq.html. Accessed: 2016-6-27.

[34] Mohammad Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *Proc. of the Thirtieth int. conf. on Very large data bases*. VLDB Endowment, 2004.

[35] Shun-Yun Hu and Guan-Ming Liao. Scalable peer-to-peer networked virtual environment. In *Proc. of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '04. ACM, 2004.

[36] Afsin Akdogan, Cyrus Shahabi, and Ugur Demiryurek. Toss-it: A cloud-based throwaway spatial index structure for dynamic location data. In *2014 IEEE 15th International Conference on Mobile Data Management*, volume 1. IEEE, 2014.

[37] Donald D Chamberlin and Raymond F Boyce. Sequel: A structured english query language. In *Proc. of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, pages 249–264. ACM, 1974.

[38] E Lai. Computerworld: No to sql anti-database movement gains steam, 2009. http://www.computerworld.com/article/2526317/database-administration/no-to-sql–anti-database-movement-gains-steam.html, 2013. Accessed: 2016-6-17.

[39] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[40] D Obasanjo. Building scalable databases: Denormalization, the nosql movement and digg. http://highscalability.com/building-scalable-databases-denormalization-nosql-movement-and-digg, 2013. Accessed: 2016-6-17.

[41] Nati Shalom. No to sql? anti-database movement gains steam–my take. http://blog.gigaspaces.com/no-to-sql-anti-database-movement-gains-steam-my-take/, 2009. Accessed: 2016-6-17.

[42] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[43] MongoDB. Foursquare. www.mongodb.com/customers/foursquare. Accessed: 2016-6-17.

[44] L Ferrucci, L Ricci, M Albano, R Baraglia, and M Mordacchini. Multidimensional range queries on hierarchical voronoi overlays. In *Journal of Computer and System Sciences*. Elsevier, 2016.

[45] NCEDC (2014). Northern california earthquake data center. uc berkeley seismological laboratory. http://www.quake.geo.berkeley.edu/anss/catalog-search.html. Accessed: 2016-6-17.

[46] Glen Van Brummelen. *Heavenly mathematics: The forgotten art of spherical trigonometry*. Princeton University Press, 2013.

[47] Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *P2P'09 - Proc. of the 9th Int. Conf. on Peer-to-Peer*. IEEE, 2009.

[48] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. Driving with knowledge from the physical world. In *Proc. of the 17th ACM SIGKDD Int. conf. on Knowledge discovery and data mining*. ACM, 2011.

[49] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. T-drive: driving directions based on taxi trajectories. In *Proc. of the 18th SIGSPATIAL Int. conf. on advances in geographic information systems*. ACM, 2010.

[50] Dublin City Council. Dublin bus gps sample data from dublin city. www.data.dublinked.ie. Accessed: 2016-6-17.

[51] Neham Mahajan. Content addressable network. github.com/neham/content-addressable-network. Accessed: 2016-6-27.

[52] John C Davis and Robert J Sampson. *Statistics and data analysis in geology*, volume 646. Wiley New York et al., 1986.