A New Spatial Database Management System Using an Hyperbolic Tree

Telesphore Tiendrebeogo
Polytechnic University of Bobo-Dioulasso
Bobo-Dioulasso, Burkina Faso
tetiendreb@gmail.com

Abstract*—Spatial information processing has been a focus of research in the past decade. In spatial databases, data are associated with spatial coordinates and extents, and are retrieved based on spatial proximity. A formidable number of spatial indexes have been proposed to facilitate spatial data retrieval. In this paper, we propose a new scalable, reliable and consistent architecture for spatial database indexing based on hyperbolic topology. This structure uses a strategy that enables to avoid such an exhaustive search that is based on the use of a centralized index. Our architecture is comparable to the well-known R-tree structure, but uses the hyperbolic geometry properties with specific model called "Poincaré disk model". It uses a distributed balanced Q-degree spatial tree that scales with spatial data objects insertions into potentially any number of storage servers through virtual hyperbolic coordinates taken in hyperbolic space. In other words, we propose a new model based on structured Distributed Hash Table (DHT) in which a user/application of each server is considered as a client node. Thus, a database server can connect to the system through a random database server that already addresses the tree with a free address given by one other database server of the system. Database servers addresses are associated with virtual hyperbolic coordinates computed by the system. In this paper, we consider a 3-regular Hyperbolic-tree in the hyperbolic plane; so, except for the root of this tree, others servers have in the best cases two addresses to give a new one (server's father uses one address). In our work, we perform simulations and we analyze the practicability and reliability compared with other DHT, such as Chord, Pastry, Kademlia used in other spatial databases. The results show that our spatial architecture based on a hyperbolic tree is viable, consistent and has acceptable performances given the scalability and flexibility it could provide to distributed database applications.

Keywords–Spatial Database; Management; Hyperbolic-Tree; Query; Storage.

I. INTRODUCTION

The typical notion of a database is that of a system in which one stores data about a fixed enterprise that one has modelled in some way. Quite often, the modelling is constrained in ways that help to reduce the complexity to a manageable level. The usual kinds of uses to which that data are put include fairly straightforward applications, like inventory management, shipping systems, and payroll systems. However, research in databases has advanced the state-of-the-art sufficiently that we are now moving into non-traditional applications that could not have been foreseen even ten years ago. This move has been largely demand-driven. For example, the emergence of spatial databases was initially driven by the demand for managing large volume of spatial data used in the geosciences and Computer-Aided Design (CAD). Also, we notice to a collection of spatial data that increasing as never seen.

Thus, in this paper, we aim at indexing large datasets of spatial objects, each uniquely identified by an Object IDentifier (OID) and stored in the hyperbolic-tree with a given address. We define a scalable and reliable index that generalizes R-tree structure for a Spatial Database System (SDS) [1]. Our architecture has conformed to the general principles of a Seamless Data Distribution System (SDDS) [2]:

- No central directory is used for data addressing,
- Servers are dynamically added to the system when needed,
- Servers address the structure through coordinates.

Our model permits redundancy of object references, like MSPastry [3], Chord [4], and Kademlia [5]. The fundamental principle of our systems is to map a large database object identifier space onto a set of servers localisation in a deterministic and distributed way. Roughly, given an object key, the system is able to obtain the locations of database servers where the corresponding values are stored. The existing issues consist mainly of how they store and look up key-value pairs of spatial data. These two primitives remain a challenging problem in distributed data despite the considerable research efforts that have been made. A spatial information system is a more general term to describe an SDS that has application customized tools for analysing spatial properties of the data. It has to model reality, integrate spatial data acquired from different sources and by different means, and support processing and analysis of data on demand. Without being exhaustive, we outline here the key issues that we address. Besides, for instance, store and lookup queries are messages forwarded in a progressive manner across cluster paths to the adequate server that contain response. Also, the underlying routing process thus requires that each database server maintains the status of its connections to other databases servers increasing drastically the number of messages exchanged, and this may constitute a severe scaling limitation. Furthermore, each database server's routing table must always be kept as small as possible in order to both reduce the store/lookup query latency and not affect the performances of applications built over our spatial database system. The same applies to the number of routing hops that must not grow too fast with the number of database servers in the system [6]. Moreover, most of spatial database systems suffer from the lack of flexibility in storage queries (i.e., values placement) involving consequently a heavy lookup traffic load on the lookup paths to the underlying servers. We argue that we can address and overcome simultaneously all the aforementioned issues by maintaining a good trade off between robustness, efficiency and system complexity. To this end, we promote an indexing system for spatial database relying on hyperbolic geometry. In this paper, we make the following contributions:

- We provide a new architecture for indexation in the distributed database system without any constraints. Thus, at first, the database servers can connect arbitrarily to each other during his phase of connection to the system. In addition, the data objects can be inserted, updated or deleted out of the system efficiently, without the cost of maintaining any global knowledge. Our approach is based on the ground breaking work of Kleinberg [7] that we have enhanced in order to manage a dynamic topology which is able to grow and shrink over time.
- We propose a specific key-based routing system for storing the mapping of database object identifier to addresses over hyperbolic system in a flexible and efficient way. Values are stored in a way to avoid overloading a particular zone of the spatial database system. Furthermore, storage and retrieving queries can be solved within $O(logN)$ hops.
- To improve database object availability and access performance, our system that is based on a Distributed Hash Table (DHT) system embeds a redundancy and caching scheme that can be parametrized to get a good trade off between reliability and storage consumption.
- We have carried out simulations, firstly, to demonstrate the interest in terms of feasibility to build a spatial database system over architecture based on the hyperbolic plane, and secondly, to compare our solution with others solutions existing based on DHT.

The remainder of this paper is organized as follows. Section II gives a brief overview of the related previous work. Section III highlights some properties of the hyperbolic plane when represented by the Poincaré disk model. Section IV defines the local addressing and greedy routing algorithms of the distributed database system. Section V defines the binding algorithm of our spatial database system. Section VI presents the results of our practicability evaluation obtained by simulations and we conclude in Section VII.

II. Related work

Spatial databases generally refer to the collection of data which have spatial coordinates and are defined within a space. Until recently, in the most of the spatial indexing design, objects in SDS are of irregular shape, and the irregular shape of these objects is the main cause for expensive spatial operator computation. Consider one of the simplest operators, such as intersection. The intersection of two polyhedra requires testing all points of one polyhedron against the other. The intersection of two polyhedra objects is not always a polyhedron; the intersection may sometimes consist of a set of polyhedra. The efficiency of these operations depends on the representation of the data [8], storage of objects [9][10] and retrieval of relevant data for computation. When data objects are stored in disks, the semantics of the data must be captured so that they can be reconstructed correctly and efficiently. Representation schemes developed for geometric modelling [3][4][5] are suitable for spatial objects. In all that precedes, they have used an Euclidean space splitting or ring, which differs from our case; we are interested in a tiling of the hyperbolic space, generally, and more particularly, to the hyperbolic plan. An elementary property of the Euclidean space is the impossibility to create more than two half planes without they intersect. Our embedding is based on the geometric property of the hyperbolic plane which allows to create distinct areas called half planes. As explained by Miquel [11], in the hyperbolic plane, we can create n half spaces pair wise disjoint whatever n. This property is the base of our embedded algorithm (red line Figure 1). Another important property is that we can tile the hyperbolic plane with polygons of any size, called p-gons. Besides the Merkle Hash Tree (MH-tree) [12] is a main-memory of Advanced Data System (ADS) that has influenced several authenticated processing techniques. It is a binary tree that hierarchically organizes hash1 values (or digests). We use the similar principle except that the key serves to calculate virtual coordinates of the hyperbolic space for more flexibility. The VB-tree [13] is a disk-based ADS that establishes the soundness, but not the completeness, in term of security of the obtained result. For keyword-based retrieval, they have integrated R-tree [14] with spatial index and signature file [15]. By combining R-tree and signature, they have developed a structure called the IR2-tree [15]. IR2-tree has merits of both R-trees and signature files. Our system, like IR2-tree, preserves objectś spatial proximity, which is important for solving spatial queries; but furthermore, it permits a reliable replication, like in Chord, MSPastry and Kademlia.

III. Hyperbolic geometry

The model that we use in our system to represent the hyperbolic plane is called the Poincaré disk model. In the Poincaré disk model, the hyperbolic plane is represented by the open unit disk of radius 1 centered at the origin. In this specific model:

- Points are represented by points within this open unit disk.
- Lines are represented by arcs of circles intersecting the disk and meeting its boundaries at right angles.

In this model, we refer to points by using complex coordinates.

An important property is that we can tile the hyperbolic plane with polygons of any sizes, called p-gons. Each tessellation is represented by a notation of the form $\{p, q\}$, where each polygon has p sides with q of them at each vertex. There exists a hyperbolic tessellation $\{p, q\}$ for every couple $\{p, q\}$ obeying $(p-2)*(q-2) > 4$. In a tiling, p is the number of sides of the polygons of the *primal* (the black edges and green vertices in Figure 1) and q is the number of sides of the polygons of the *dual* (the red triangles in Figure 1). Our purpose is to partition the plane and address each node uniquely. We set p to infinity, thus transforming the primal into a regular tree of degree q. The dual is then tessellated with an infinite number of q-gons. This particular tiling splits the hyperbolic plane in distinct spaces and constructs an embedded tree that we use to assign unique addresses to the nodes. An example of such a hyperbolic tree with $q = 3$ is shown in Figure 1.

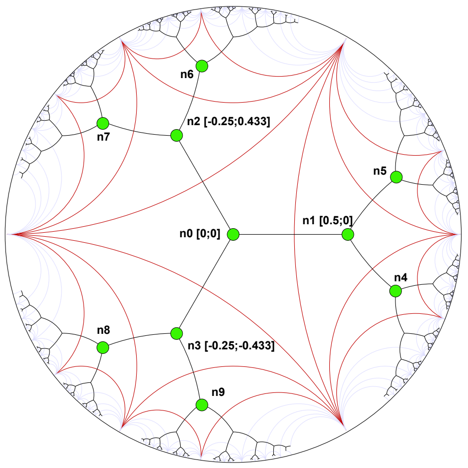

Figure 1. 3-regular tree in the hyperbolic plane.

In the Poincaré disk model, the distances between any two points z and w are given by curves minimizing the distance between these two points and are called geodesics of the hyperbolic plane. To compute the length of a geodesic between two points z and w and thus obtain their hyperbolic distance $d_{\mathbb{H}}$, we use the Poincaré metric which is an isometric invariant given by the formula:

$$d_{\mathbb{H}}(z, w) = argcosh(1 + 2 \times \frac{|z - w|^2}{(1 - |z|^2)(1 - |w|^2)}) \quad (1)$$

This formula is used by the greedy routing algorithm shown in the next section.

IV. Naming and Binding Technical in the Hyperbolic Plane

We now explain how we create the hyperbolic addressing tree for spatial database servers joins and how queries can be routed in our SDS. We propose here a dynamic and scalable hyperbolic greedy routing algorithm [16]. The first step in the creation of our SDS based on hyperbolic-tree of servers nodes is to start the first database server and to choose the degree of the addressing tree.

We recall that the hyperbolic coordinates (i.e., a complex number) of a server node of the addressing tree are used as the address of the corresponding database server in the SDS. A server node of the tree can give the addresses corresponding to its children in the hyperbolic-tree. The degree determines how many addresses each database server will be able to give for news nodes servers connexions. The degree of the hyperbolic-tree is defined at the beginning for all the lifetime of the SDS. The SDS is then built incrementally, with each new data server joining one or more existing data servers. Over time, the data servers will leave the overlay until there is no server left which is the end of the SDS. So, for every data object that must be stored in the system, an OID is associated with it and map then in key-value pair. The key will allow to determine in which data servers the object will be stored (like in the Section V). Furthermore, when a data object is deleted, the system must be able to update this operation in all the system by forwarding query through the latter. This method is scalable because unlike [17], we do not have to make a two-pass algorithm over the whole spatial system to find its highest degree. Also, in our system, a server can connect to any other server at any time in order to obtain an address.

The first step is thus to define the degree of the tree because it allows building the *dual*, namely the regular $q - gon$. We nail the root of the tree at the origin of the *primal* and we begin the tiling at the origin of the disk in function of q. Each splitting of the space in order to create disjoint subspaces is ensured once the half spaces are tangent; hence, the *primal* is an infinite q-regular tree. We use the theoretical infinite q-regular tree to construct the greedy embedding of our q-regular tree. So, the regular degree of the tree is the number of sides of the polygon used to build the *dual* (see Figure 1). In other words, the space is allocated for q child database servers. Each database server repeats the computation for its own half space. In half space, the space is again allocated for $q - 1$ children. Each child can distribute its addresses in its half space. Algorithm 1 shows how to compute the addresses that can be given to the children of a database server. The first database server takes the hyperbolic address (0;0) and is the root of the tree. The root can assign q addresses.

Algorithm 1 Calculating the coordinates of a database server's children.

```
 1: procedure CALCCHILDRENCOORDS($server$, $q$)
 2:     $step \leftarrow argcosh(1/sin(\pi/q))$
 3:     $angle \leftarrow 2\pi/q$
 4:     $childCoords \leftarrow server.Coords$
 5:     for $i \leftarrow 1, q$ do
 6:         $ChildCoords.rotationLeft(angle)$
 7:         $ChildCoords.translation(step)$
 8:         $ChildCoords.rotationRight(\pi)$
 9:         if $ChildCoords \neq server.ParentCoords$ then
10:             STORECHILDCOORDS($ChildCoords$)
11:         end if
12:     end for
13: end procedure
```

This distributed algorithm ensures that the database servers are contained in distinct spaces and have unique coordinates. All the steps of the presented algorithm are suitable for distributed and asynchronous computation. This algorithm allows the assignment of addresses as coordinates in dynamic topologies. As the global knowledge of the SDS is not necessary, a new server can obtain coordinates simply by asking an existing server to be its parent and to give it an address for itself. If the asked server has already given all its addresses, the new server must ask an address to another existing database server. When a new server obtains an address, it computes the addresses (i.e., hyperbolic coordinates) of its future children(The new database servers which shall connect to the SDS). The addressing hyperbolic-tree is thus incrementally built at the same time than the SDS.

When a new database server has connected to database servers already inside the SDS and has obtained an address from one of those database servers, it can start sending requests to store or lookup database object in the SDS. The routing process is done on each database server on the path (starting from the sender) by using the greedy algorithm 2 based on the hyperbolic distances between the servers. When a query is received by a database server, the database server computes the distance from each of its neighbors to the destination and forwards the query to its neighbor which is the closest to the destination (destination database server computing is given in the Section V). If no neighbor is closer than the server itself, the query has reached a local minima and is dropped.

Algorithm 2 Routing a query in the distributed database system.

```
1: function GETNEXTHOP($dataserver$, $query$) return dataserver
2:     $w = query.destinationServerCoords$
3:     $m = server.Coords$
4:     $d_{min} = argcosh\left(1 + 2\frac{|m-w|^2}{(1-|m|^2)(1-|w|^2)}\right)$
5:     $p_{min} = server$
6:     for all $neighbor \in server.Neighbors$ do
7:         $n = neighbor.Coords$
8:         $d = argcosh\left(1 + 2\frac{|n-w|^2}{(1-|n|^2)(1-|w|^2)}\right)$
9:         if $d < d_{min}$ then
10:            $d_{min} = d$
11:            $p_{min} = neighbor$
12:        end if
13:    end for
14:    return $p_{min}$
15: end function
```

In a real network environment, link and server failures are expected to happen often. If the addressing hyperbolic-tree is broken by the failure of a database server or link, we flush the addresses attributed to the servers beyond the failed server or link and reassign new addresses to those servers (some servers may have first to reconnect to other servers in order to restore connectivity). But, this solution is not developed in this paper.

V. Hyperbolic adressing and destination spatial database servers computing

In this section, we explain how our SDS computes the destination database servers addresses for stores and retrieves queries. Indeed, the first server contacted by a client (prime server) for sending a query in the system considers this latter as a data object that can be stored or looked up. Thus, this server generates an OID associate to the data object and this latter is mapped into hyperbolic addresses corresponding to destination database servers addresses in the hyperbolic-tree. Our solution of SDS is a structured DHT system that uses the local addressing and the greedy routing algorithms presented in Section IV. On start-up, each new client query is associated with the data object with OID corresponding to the name of the query and that identifies the query it runs on. This name will be kept by the data object during all the lifetime of the SDS. When the prime database server computes some specific addresses of database servers, when it is about a storage query, it stores the name (OID) and value of query in these specific addresses of SDS, thus the data object in the DHT; when it is about a retrieving query, it contacts database servers whose addresses have been computed. In our spatial system, the name is used as a key by a mathematical transformation. If the same name is already stored in the SDS, an error message is sent back to the prime server (Server to whom the client is directly bound) in order to generate another name. Thus, the SDS structure itself ensures that OIDs are unique.

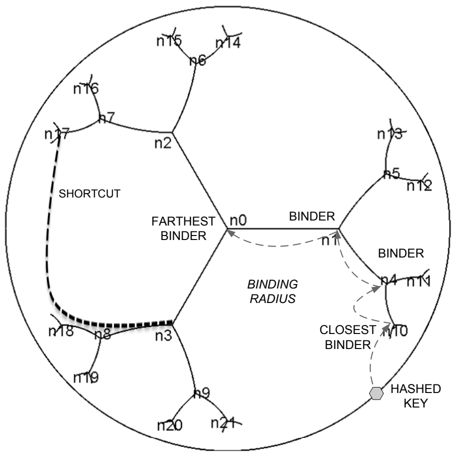

Figure 2. Hyperbolic DHT system.

A (OID, value) pair, with the name acting as a key by mapping is called a *binding*. Figure 2 shows how and where a given binding is stored in the SDS. A binder is any database server that stores these pairs. The depth of a server in the addressing hyperbolic-tree is defined as the number of parent servers to go through for reaching the root of the hyperbolic-tree (including the root itself). When the distributed database system is created, a maximum depth for the potential binders is chosen. This value is defined as the *binding hyperbolic-tree depth*. To ensure a load balancing of the system, the depth d is chosen such d minimize the following in-equation :

$$p \times (\frac{(1-(p-1)^d)}{2-p}) + 1 \geq N \qquad (2)$$

with d the depth, p ($p \geq 3$) the degree and N the number of database servers.

When a new database server joins the SDS by connecting to other servers, it obtains an address from one of these servers. Next, the server stores its own binding in the system. So, during his life, each database server tries to join others by sending a join query. Each server cannot accept that a limited number of join queries independently of the degree of the Hyperbolic-tree. The new connections serve as shortcuts during the phases of storage and retrieving of data objects. We call these connections, shortened links, as indicated in Figure 2.

A. Storage query process

When a client wants to send a storage query (insert, etc.), the first server with whom it is connected considers a query as an object (thus generates an OID) and creates a key by hashing its name with the SHA-512 algorithm. It divides the 512-bit key into 16 equally sized 32-bit sub keys (for redundant storage). The server selects the first sub key and maps it to an angle by a linear transformation.

The angle is given by:

$$\alpha = 2\pi \times \frac{\texttt{32-bit subkey}}{\texttt{0xFFFFFFFF}} \tag{3}$$

The database server then computes a virtual point v on the unit circle by using this angle:

$$v(x, y) \text{ with} \begin{cases} x = cos(\alpha) \\ y = sin(\alpha) \end{cases} \tag{4}$$

Next, the database server determines the coordinates of the closest binder to the computed virtual point above by using the given *binding tree depth.*

In Figure 2, we set the *binding Hyperbolic-tree depth* to three to avoid cluttering the figure. It is important to note that this closest binder may not really exist if no database server is currently owning this address. The database server then sends a storage query to this closest database server. This query is routed inside the SDS by using the greedy algorithm of the Section IV. If the query fails because the binder does not exist or because of database server/link failures, it is redirected to the next closest binder which is the father of the computed binder. This process continues until the query reaches an existing binder database server which can be any database server on the path from the computed closest binder to the center database server.

Upon reaching an existing binder, the query is stored in that binder. The query can thus go up the addressing Hyperbolic-tree to the center database server having the address (0;0) which is the farthest binder. The path from the computed closest binder to the farthest binder is defined as the binding radius. This process ensures that the queries are always stored first in the binders closer to the unit circle and last in the binders closer to the disk center. However, to avoid overloading the farthest binder particularly and to keep a load balancing, we limit the number of storages S like follow:

$$S \leq \lfloor \frac{1}{2} \times \frac{log(N)}{log(q)} \rfloor \tag{5}$$

with N equal to number of distributed database servers, q to degree of hyperbolic-tree.

Furthermore, if addressing hyperbolic-tree is unbalanced (because of servers leaving or failing in the system), many queries may be stored in database servers close to the center thus overloading them. Besides the previous solution, any binder will be able to set a maximum number of stored queries and any new database server to store will be refused and the query redirected as above. Besides, to provide redundancy and so ensure the availability and reduce the latency period in the lookup process, the database server does the storage process described above for each of the other 15 sub keys. Thus 16 differents binding radiuses will be used at the most and this will improve the even distribution of the pairs (key-value).

In addition to this and still for redundancy purposes, a pair key-value of the data object may be stored in more than one database server of the binding radius. A binder could store a data object and still redirect its query for storage in other ancestor binders. The number of stored copies of a key-value pair along the binding radius may be an arbitrary value set at the SDS creation. Similarly, the division of the key in 16 sub keys is arbitrary and could be increased or reduced depending on the redundancy needed. To conclude, we can define two redundancy mechanisms for storage copies of a given binding:

1) We can use more than one binding radius by creating several uniformly distributed subkeys.
2) We can store the data object key-value pair in more than one binder in the same binding radius.

Algorithm 3 Storage algorithm in the general context

```
1: function StoreProcess($PrimeDataServer\ \ Query$) return 0
2:     $OID \leftarrow Query.GetOID()$
3:     $Key \leftarrow Hash(OID)$
4:     for A doll $r \in R_{Circular}$
5:         $d \leftarrow P_{Max}$
6:         $i \leftarrow 1$
7:         while $i \leq \left\lfloor \frac{1}{2} \times \frac{log(N)}{log(q)} \right\rfloor$ && $d \geq 0$ do
8:             $SubKey[r][d] \leftarrow C_Subkey(Key)[r][d]$
9:             $TgServAd[r][d] \leftarrow C_Ad(SubKey[r][d])$
10:            $TgServ \leftarrow GetTg(TgServAd[r][d])$
11:            if $route(Query, TgServer)$ then
12:                $i++$
13:                $put(OID, Query)$
14:            end if
15:            $d--$
16:        end while
17:    end for
18:    return 0
19: end function
```

These mechanisms enable our SDS to cope with a non-uniform growth of the database servers and they ensure that a data object will be stored in a redundant way that will maximize the success rate of its retrieval.

The numbers of sub keys and the numbers of copies in a radius are parameters that can be set at the creation of the SDS. Increasing them leads to a trade-off between improved reliability and lost storage space in binders.

Our solution has the property of consistent hashing: if one database server fails, only its keys are lost but the other binders are not impacted and the whole system remains coherent.

As in many existing systems, pairs (key-value) will be stored by following a hybrid soft and the hard state strategy. Every database server has to verify every X unit of time that his neighbour is alive. When the database server leaves the system, his neighbor tries to connect with his father for keeping the hyperbolic-tree stable. But, we do not detail this dynamic process in our paper. Algorithm 3 illustrates this mechanism.

B. Lookup query process

Now, if the client wants to lookup a data object in the connected and use to store the data objects distributed SDS, a prime server is contacted and this latter generates an OID for the client query. Here again, OID is mapped into a key by SHA-512 algorithm, thus the 512 bits key is divided into 16 sub keys. Each sub key by the process describe in the Section V-A, will be transform into address that represent address of the database server where data object is stored. This latter is contacted by prime database server for update, delete or select the value associated.

Algorithm 4 Lookup algorithm in general context for inserting, deleting and updating of data object

```
1: function LookupProcess($PrimeDataServer, Query$) return $Value$
2:     $OID \leftarrow Tg.GetQueryOID()$
3:     $Key \leftarrow Hash(QueryOID)$
4:     for A doll $r \in R_{Circulare}$
5:         $d \leftarrow P_{Max}$
6:         $i \leftarrow 1$
7:         while $i \leq \left\lfloor \frac{1}{2} \times \frac{log(N)}{log(q)} \right\rfloor$ $\&\&$ $d \geq 0$ do
8:             $TgServAd[r][d] \leftarrow GetValue(Key)$
9:             $Value \leftarrow GetValue(TgSerAd[r][d], OID)$
10:            if $Value\ != null$ then
11:                if $Query\ == delete$ then
12:                    $delete(OID)$
13:                end if
14:                if $(Query\ == update)$ then
15:                    $update(OID)$
16:                end if
17:                if $Query\ == select$ then
18:                    return $Value$
19:                    $break$
20:                end if
21:                $i++$
22:            end if
23:            $d--$
24:        end while
25:    end for
26:    return 0
27: end function
```

When the redundancy mechanism has been used to store the data object, lookup process repeats the latter process of lookup for any sub key, thus, the operation will be performed on all database servers that contain the data object. Our SDS ensures then the coherence of data objects of the spatial database. The Algorithm 4 illustrates this mechanism.

VI. Experimental evaluation

We performed experiments evaluating the behaviour of our SDS over large datasets of an hyperbolic plane. Furthermore, we consider that the system is dynamic (there is join or leave of database servers during the simulation) with rates of churn variables. We use a simulator Peersim [18] for SDS simulation and it allows to obtain dataset OIDs by generation following the uniform distribution. The study involved the following parameters of our SDS :

- number of database servers connected and used to store the distributed data objects. Here we have considered 10000 at the beginning;
- we try to store 6 million data objects in our SDS following an exponential distribution with a median equal to 10 minutes;
- we perform simulation during 2 hours and, we fixed the capacity of each server to 6000 objects.
- we consider that the rate of churn varies between 10% and 60%.

We studied the behaviour of our architecture for the data objects storage and retrieving in the system. So, we are interested by certain properties.

A. Balance of the Hyperbolic-tree

Figure 3 shows an experimental distribution of points corresponding to the scatter plot of the distribution of database server in our system. Thus, we can mark that our hyperbolic-tree is balanced. Indeed, we can noticed from part and others around the unit circle which we have database servers. This has an almost uniform distribution around the root, what implies that our system builds a well-balanced tree what will more easily allow to reach a load balancing of storage.

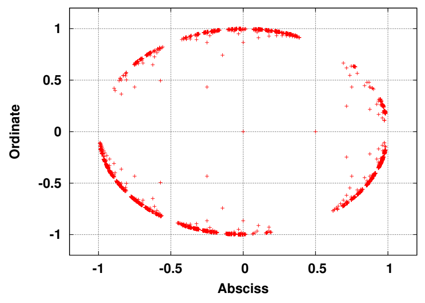

Figure 3. Scatter plot corresponding to the distributed database servers.

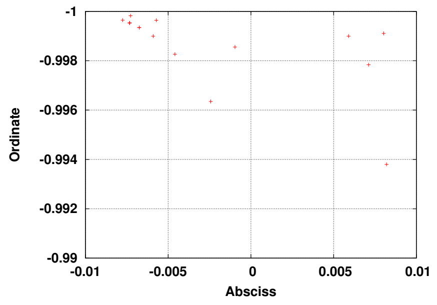

Figure 4. Distribution of database servers in the neighborhood on the edge of the unit circle.

Figure 4 shows correspondingly Poincaré disk model that no address of database server belongs on the edge of the unit circle. Indeed, the addresses of database server were obtained by projection of the tree of the hyperbolic plane in a circle of the Euclidean plan of radius1 and of center of coordinate (0; 0).

The distances between any two points z and w in the Poincaré disk model given in equation 6 can also be written like follows :

$$d_{\mathbb{H}}(z,w) = argcosh(1+2\Delta) \tag{6}$$

with:

$$\Delta = \frac{|z-w|^2}{(1-|z|^2)(1-|w|^2)} \tag{7}$$

For more details on the Poincaré metric, we refer the reader to the proof in [19]. The hyperbolic distance $d_{\mathbb{H}}(z,w)$ is additive along geodesics and is a Riemannian metric. Thus, we consider the formulae 6, the distance between O (0;0) and any point on edge to the circle so much towards the infinite. All the computed addresses associated to the database servers should be inside the unit circle. This result shows that our distributed database system can grow in the infinite, in theory. In practice of other parameters than we have not evoke in this paper have to intervene. Our system, besides ensuring the security of the data object by hashing them, the availability by the replication of the storage so the coherence of the data object by the lookup for all the occurrences of an OID for the update, the deletion, allows a passage to the scale in term of the number of database servers interconnected in the system.

B. Load balancing of the Hyperbolic-tree

Figure 5 shows a plot of the evaluation of the average number of objects stored by database server in time. This figure shows a regular growth of this number of data objects stored in function of time. Indeed, 293.27 data objects on average are stored by database server after 10 minutes vs 620.4 after 2 hours. What is interesting to notice is that the standard deviations seem low, approximately 10 % of the average. This indicates a low dispersal of the number of objects stored around of average during the simulation.

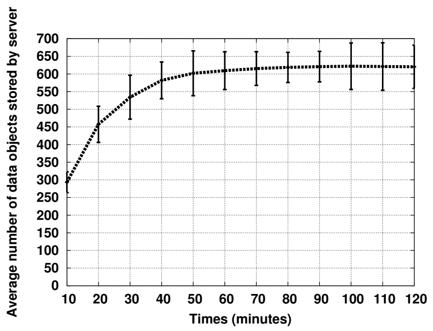

Figure 5. Distribution of the spatial database servers on the hyperbolic-tree.

Indeed, if we use our results to build the confidence interval, we can say that after 10 minutes of simulation, 68.2 % of the database servers stores between 263.69 and 322.71 data objects and 95 % stores it between 234.18 and 352.22 against 68.2 % of the database servers who stores between 560.18 and 681.58 data objects after 2 hours and 95 % of the database servers who stores between 497.95 and 742.84 data objects after 2 hours. In view of these , we can say that our system maintains a good enough load balancing between database servers which is to ensure the stability of our SDS.

C. Storage and retrieval path length in Hyperbolic-tree

Figures 6 and 7 show that during the simulation, queries in both cases can be answered within O (log N) or N equal to the number of database server building the system. Indeed, the standard deviation being very low (less than 5 % of the average for storage et retrieving), we did not represent it. In the worst case, queries contacts less than 4 database servers in a system which in account 10000, for either to store, or to retrieving a data object.

Besides, what is also interesting to note is that the plot decreases slowly to become stationary after around 100 minutes in both cases. It can be explained because during the simulation, the database servers create shortcuts as indicated in the Section V. These shortcuts allow to reach their target in fewer hops. The situation of stationarity is understandable by the fact that after a while, all the database servers reached their maximum of shortcuts create and most part of the queries is processed on average in less than 3.75 hops in both cases.

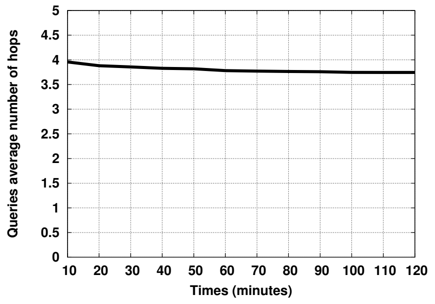

Figure 6. Path length of the storage's queries in Hyperbolic-tree.

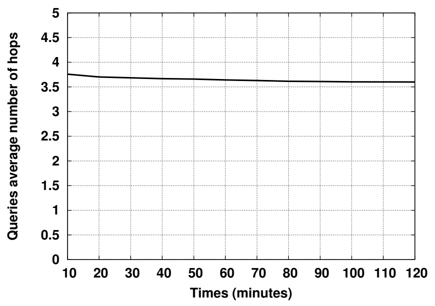

Figure 7. Path length of the lookup's queries in Hyperbolic-tree.

D. Analyses compared by the performances

Figure 8 shows that in a context where churn phenomenon is equal to 10%, we can notice that all the successes rates are between 83% and 88% when the churn rate becomes 60%, according to the number of replication, the success rates is between 18% and 67%. This result indicates that the replication strategy permits to reduce impact of the churn phenomenon on our spatial database performance.

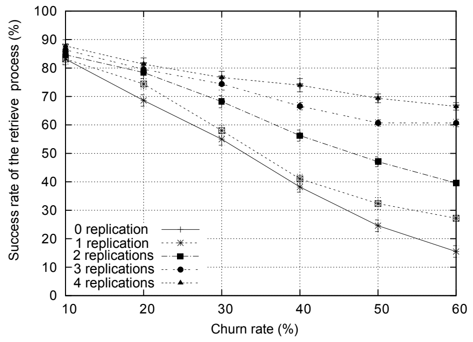

Figure 8. Evaluation of resilient strategy against the churn phenomenon.

Figure 9 indicates that when the churn rate is between 10% and 60%, the average number of hops to send query from source server to target server (server that contains the data object searched) is approximatively the same. Thus, through this result, we show that our system behaves well with compared with the others such as Chord, MSPastry and Kademlia. So, our spatial database that is based on this DHT algorithm has a good performance.

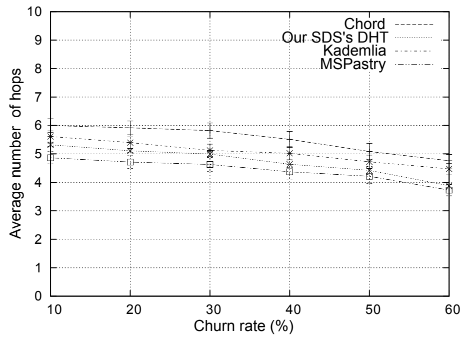

Figure 9. Comparison of the success rate depending to DHTs algorithms.

VII. Conclusion and future work

In this paper, we presented a new structure based on the Poincaré disk model. The hyperbolic tree which is used presents some properties that allow us to propose a consistent system of distributed database servers using the virtual address (hyperbolic coordinates). Next, we evaluated the performances of our system according to some parameters.

We showed that our system was scalable in terms of the number of database servers that we can connect as well as in term of number of hops to solve the queries. We also showed that the arrangement of the different database servers of the model allows us to keep a well-balanced tree. Furthermore, we showed that our system maintains a storage load balancing. Furthermore, this model based on Disk of Poincaré present a certain number of properties that permit him to be more consistent and hardness than other existing solution. All these results are encouraging and show us that our system is viable in a dynamic context. In the continuation of our work, we envisage to study our SDS in a realistic context and improve on Churn-resilient replication strategy.

References

[1] R. H. Güting, “An introduction to spatial database systems,” The VLDB Journal, vol. 3, no. 4, Oct. 1994, pp. 357–399.

[2] S. Jajodia, W. Litwin, and T. J. E. Schwarz, “Lh*re: A scalable distributed data structure with recoverable encryption.” IEEE, 2010, pp. 354–361.

[3] M. Castro, M. Costa, and A. I. T. Rowstron, “Performance and dependability of structured peer-to-peer overlays,” in 2004 International Conference on Dependable Systems and Networks (DSN 2004), 28 June - 1 July 2004, Florence, Italy, Proceedings, 2004, pp. 9–18.

[4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications.” New York, NY, USA: ACM, 2001, pp. 149–160.

[5] P. Maymounkov and D. Mazières, “Kademlia: A peer-to-peer information system based on the xor metric.” London, UK, UK: Springer-Verlag, 2002, pp. 53–65.

[6] S. A. Idowu and S. O. Maitanmi, “Transactions- distributed database systems: Issues and challenges,” IJACSCE., vol. 2, no. 1, Mar. 2014.

[7] R. Kleinberg, “Geographic routing using hyperbolic space,” in in Proceedings of the 26th Annual Joint Conference of INFOCOM. IEEE Computer and Communications Societies, 2007, pp. 1902–1909.

[8] O. Günther, Efficient Structures for Geometric Data Management, ser. Lecture Notes in Computer Science. Berlin / Heidelberg / New York: Springer-Verlag, 1988, vol. 337.

[9] H. Lu, B. C. Ooi, A. DŠouza, and C. C. Low, “Storage management in geographic information systems,” ser. Lecture Notes in Computer Science, vol. 525. Springer, 1991, pp. 451–470.

[10] T. Tiendrebeogo, D. Ahmat, and D. Magoni, “Reliable and scalable distributed hash tables harnessing hyperbolic coordinates,” in NTMS’12, 2012, pp. 1–6.

[11] A. Miquel, “Un afficheur générique d’arbres à l’aide de la géométrie hyperbolique,” in In Journées francophones des langages applicatifs (JFLA), 2000, pp. 49–62.

[12] R. C. Merkle, “A certified digital signature,” in Proceedings on Advances in Cryptology, ser. CRYPTO ’89. New York, NY, USA: Springer-Verlag New York, Inc., 1989, pp. 218–238. [Online]. Available: http://dl.acm.org/citation.cfm?id=118209.118230

[13] H. Pang and K.-L. Tan, “Authenticating query results in edge computing,” in Proceedings of the 20th International Conference on Data Engineering, ser. ICDE ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 560–. [Online]. Available: http://dl.acm.org/citation.cfm?id=977401.978163

[14] A. Guttman, “R-trees: A dynamic index structure for spatial searching,” SIGMOD Rec., vol. 14, no. 2, Jun. 1984, pp. 47–57. [Online]. Available: http://doi.acm.org/10.1145/971697.602266

[15] G. R. Hjaltason and H. Samet, “Distance browsing in spatial databases,” ACM Trans. Database Syst., vol. 24, no. 2, Jun. 1999, pp. 265–318. [Online]. Available: http://doi.acm.org/10.1145/320248.320255

[16] F. Papadopoulos, D. Krioukov, M. Boguñá, and A. Vahdat, “Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces,” in Proceedings of the 29th Conference on Information Communications, ser. INFOCOM’10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 2973–2981. [Online]. Available: http://dl.acm.org/citation.cfm?id=1833515.1833893

[17] V. Gaede and O. Günther, “Multidimensional access methods,” ACM Comput. Surv., vol. 30, no. 2, Jun. 1998, pp. 170–231. [Online]. Available: http://doi.acm.org/10.1145/280277.280279

[18] I. Kazmi and S. F. Y. Bukhari, “Peersim: An efficient & scalable testbed for heterogeneous cluster-based p2p network protocols.” Washington, DC, USA: IEEE Computer Society, 2011, pp. 420–425. [Online]. Available: http://dx.doi.org/10.1109/UKSIM.2011.86

[19] A. F. Beardon and D. Minda, “The hyperbolic metric and geometric function theory,” vol. 956. International Workshop on Quasiconformal Mappings And Their Applications, 2007, pp. 9–57.