On Indexing Multidimensional Data In A P2P Context

Florin Dragan, Georges Gardarin, Benjamin Nguyen, Laurent Yeh
Universite de Versailles Saint-Quentin-en-Yvelines
Laboratoire PRiSM

Abstract
Based on their remarkable properties (fault tolerance, scalability, decentralization), P2P systems tend to be largely accepted as a common support for deploying massively distributed data management applications. Some of the existing P2P systems are built over hybrid networks where certain peers are specialized in indexing. Others make use of DHTs as common support for data indexing. However, only a few solutions have been proposed to solve the problem of multidimensional data indexing. Many of these suffer from severe limitations regarding load balancing and scalability of the data indexed (data pieces with many dimensions are not well indexed). In this paper we propose a new approach for indexing multidimensional data using a P2P architecture. It is based on a fault tolerant and query optimal overlay network built from a skip-graph. We index data pieces of many dimensions as vectors and we adopt the polyhedral algebra to cluster values with similar properties. For a continuous distribution of values to peers, we use a hierarchical overlay network (that we call routing layer). At the end of the paper, we offer an exhaustive evaluation that compares the main properties of our architecture against similar systems.

1 Introduction
While first used for simple queries, such as searching for filenames (or document names), Peer-to-peer (P2P) networks represent a possible basis for implementing large scale database systems. Among the main qualities that distinguish P2P networks, we stress: robustness, scalability, reliability, no central administration, no control over data placement. However one of the main problems of P2P data sharing systems resides in their weak storage and retrieval capabilities when faced with anything more complex than simple values or strings. A recent problem that attracts the attention of database communities is how P2P networks can be used for efficient management of complex data, such as XML collections, image descriptors or geographical data. It makes sense to observe that all these formats share a common property: the fact that each data unit is described by a set of values. We believe that by finding a good solution to this problem, the P2P paradigm may be totally adopted in the design and development of new database applications.

In this paper we adopt multidimensional data as the fundamental data organization format. More precisely we use the vector-based notation to represent multi-numerical-valued data. We consider that the vector-based notation is general enough to be used in complex data representation. As an example, consider that XML paths can be mapped to a vector (in the mathematical sense) by using a hash function. The XML path "/bib/book/author"
might be mapped to the vector $< \text{hash(bib)}, \text{hash(book)}, \text{hash(author)} > = < 12, 14, 3 >$. To clarify, consider the following XML fragment:

```
< bib >
  < book year = "1994" >
    < title > TCP/IP Illustrated </title >
    < author >< last > Stevens < /last >
      < first > W. < /first >=
      < publisher > Addison − Wesley < /publisher >
      < price > 65.95 < /price >
  </ book >
</ bib >
```

This XML fragment can be decomposed a set of XML paths. Among the generated paths we have: $P_1 = /\text{bib/book/year}/"1994"$  
$P_2 = /\text{bib/book/year/title}/"TCP/IP." … "$  
$P_3 = /\text{bib/book/author/last}/"Stevens"$

Each of the previous paths can be mapped to a vector of length 6. For the first path we have: $< \text{hash("bib")}, \text{hash("book")}, \text{hash("year")}, \text{hash("1994")}, \text{NIL} >$. Once encoded as a vector, each path can be indexed in the network together with document localization information. As in the existing DHT-based applications, the hashing function is shared by the entire network participants. When a vector is completely specified (i.e., the values for each dimension are given), it can be viewed as a point in an N-dimensional space, N representing the number of values in the vector. Moreover the vector-based notation can be natively used in indexing applications (e.g., applications that index image descriptors).

Most vector-based applications can perform different tasks based on the complex retrieval capabilities of a large scale vector-based indexing system. We can imagine different kinds of vector queries like single point retrieval, wild-card queries or finding all points in a certain range (range queries). As an applicative example, consider a mobile phone company that identifies each mobile call by the following parameters: the phone number (N), the localization of the mobile phone on a given bi-dimensional map (X and Y) and the duration of the call (T). In a specific application a call may be represented by the following vector: $< N, X, Y, T >$ . To generate reports the employees of the phone company may issue queries having the following form: "find all calls during the period xxx spatially located in a box given by the rectangular coordinates (10,23) (233,222)". To execute this query, we need a system capable of answering range queries over multiple axis.

To index multidimensional data, a first approach is to concatenate all values of a vector, then use a hashing function to transform the result into a single value, and finally index it. This approach is good for indexing vectors, but it can not resolve range queries, or queries involving only some of the dimensions. To preserve sub-parts of vector values, most existing proposals for multi-values indexing map each vector to a single dimension using well known data mapping techniques like Z-Order [10] or Peano-Key [11]. The principle of these mapping techniques is to encode a zone that best encloses a multi-dimensional point to be indexed. For example, consider the mapping of a bi-dimensional value to a mono-value (figure 1). As it can be observed the bi-dimensional domain is first divided into four sub-zones. For each sub-zone, a zone-code is assigned according to a Z-like traversal technique. Each point taking part in one of the zones is described by the zone specific code. In the case where another point is mapped to the same sub-zone, a new (sub-)zone division takes place. Such mapping algorithms are usually called Space Filling Curve (SFC) because they fill the space in a precise order that can be generalized for any dimension. The number of bits required
for mapping depends on the number of dimensions of the values to be indexed: in a three-dimensional space, three bits are needed for the first space-division. More generally, for a point in a real space of D dimensions, we need D bits to encode the localisation in a sub-zone at each level of the sub-division. The mapping value (routing indices) can then be used for indexing multidimensional values in a classical model of P2P network (like Chord [7] or Skip-Graph [8]).

One important limitation of such systems is that they are not adapted to concentrated (non uniform) value distributions. When most of the values are concentrated in a reduced zone of the value domain, the SFC methods require good indices precision (lots of bits) to represent the differences between two values.

Several solutions have been proposed to resolve this limitation [3][14][15]. The basic idea is to replace the traditional SFC static splitting with a dynamic generation of the best splitting boundary according to the distribution of points into sub-zones. However this solution also presents several limitations in the P2P context, since the new splitting boundaries must be broadcast over the P2P network so that other peers can perform the same mappings. Not only does this generate extra network traffic, but this approach also contradicts with the main P2P concept of maximal peer independence.

Another problem of the presented technique is that it does not scale with the multidimensional size of the values to be indexed. Indeed as the number of bits used for mapping depends on the number of dimensions of the values to be mapped, one can easily understand that a huge number of bits are required for mapping values with multiple dimensions in a zone with high density. For example, with an indice length of 24 bits we can map a value with 20 dimensions in a single level. On the contrary 12 levels may be generated for the mapping of a bi-dimensional value. In an extreme case, for a point in a very high dimension space, we are unable to map data to an indice, because the number of bits is insufficient. This clearly shows that given a number of bits n, the SFC coding technique has a precision to locate a point in the space of values, inversely proportional to the dimension of vectors to be index. In other
words, the precision bits for a zone-code in a routing indice must be shared for each value of the vector.

Another approach of multi-dimension indexing is that of Mercury [1] where a Chord structure is allocated for each dimension (figure 2). The first Chord layer indexes the first dimension and references another Chord that indexes the second dimension and so on. The method makes it possible to index vectors of very high dimension. No update information is exchanged between peers. However, the problem it is that there are as many Chord structures as number of dimensions. Moreover, measures in [1][2][3][4] show that Chords in the lower levels have a low loading rate.

To our knowledge there are very few articles in the peer to peer context that deal with the problem of indexing values of multiple dimensions (lengthy vectors) in an efficient way. From our experience in the P2P context, not being able to index high dimensional vectors is a great limitation for many applications. For example the finality of our work is to be able to index lengthy paths in a reliable P2P network without any local administration.

Our main contribution is the proposition of a P2P architecture suitable for indexing high dimension vectors that:

- Is not sensitive to the dimensions of the vectors (as shown in the section on experimental evaluation).
- Does not require the sharing of large quantities of data between the peers to achieve load balancing over the network (measurements show the number of peers involved in the process of organization is limited to a maximum of 5 for each architecture).

Compared to others works, one important specificity and originality consists in introducing the polyhedral algebra to the indexing and querying processes.

Our solution is based on a two layer P2P architecture:

1. The first layer is represented by a model of a P2P network (called routing layer) organized according to the main idea of a skip-graph. This structure provides almost the same features as other similar techniques like Chord (no centralized administration, routing in \( \log n \), etc), but it routes on a vector instead of a simple value. Moreover, it offers a greater resistance to failure (subject not developed in this paper) due to skip-graph specific algorithms. We chose the skip-graph model based on its capability of interconnecting peers that positively influences the query routing process.

2. The second layer is represented by a tree-like architecture that uses integer numbers’ ordering for routing. To be able to adopt skip-graph in our context, we must obtain the same order property for multidimensional vectors. To achieve this goal, we have integrated the polyhedral theory into the skip-graph: the second layer of our architecture is a network used for the management of the multidimensional values to be indexed based on the polyhedral theory. The main purpose of this second network layer is to control the network evolution, the (balanced) distribution of values on peers and to compute the specific intervals of zones. One important property of this layer is that it is entirely autonomous and does not require any central administration, in line with P2P philosophy.
Like other systems, our architecture can route on point-values (a vector), on ranges, but it can also route partially specified queries.

As presented in the section with experimental evaluation, our network model is insensitive to the dimension of the vectors, which encourages the implementation of a large range of P2P applications that use data pieces characterized by multiple dimensions. The query routing process is independent of the dimension of the vector and is performed, as in mono-value indexing architectures, in log n. We also show that an index update involves the participation of a limited number of peers and triggers no data exchanges (as in Chord). Moreover we show that the load is well distributed over peers.

The remainder of paper is organized as follows: Section 2 describes the first layer (routing layer) of the P2P network architecture. The second layer (data management layer) is presented in the third section. Section 4 describes the overall network architecture composed of the previously presented layers. It shows how both networks collaborate for data indexing and query routing, and how no central administration is required. Section 5 presents the experimental evaluation that demonstrates the nice properties of our network. Although there exist very few similar works, section 6 presents the main characteristics of our system compared to other systems and finally, section 7 concludes.

2 The routing layer

We propose a network overlay composed of two independent layers that collaborate for routing queries to peers that contain relevant data. One layer is inspired from the structure of skip-lists or skip-graphs and deals with query routing. We refer to it as the routing layer. The other layer has a tree-like topology and contributes to organising data in the network. It insures the splitting of the value domain in an ordered sequence of zones which determine a continuous zone of values. It is referred to as the indexing layer. By zone we denote a collection of multidimensional values described by a set of inequations as presented in the following paragraphs. In this section, we present the architecture of the routing layer. This overlay is only used for query routing. Its comprehension assumes the knowledge of the skip-graph principles and of basic polyhedral theory. In this section, we consider the routing layer as an autonomous working network and we focus on the description of the basic architecture and query routing process. Several assumed hypothesis (e.g. network evolution, zone division computation) will be proven in section 3 where we present the indexing layer.

![Figure 3: Example of a Skip Graph](image)

2.1 Background

Skip Graph basics

Skip-Graphs are completely distributed data structures based on skip-lists initially used for implementing in-memory dictionaries. A skip-graph is composed of several levels of ordered lists as illustrated in figure. The peers are ordered according to the value of the key.
(identical to routing indices, and seen as a sequence of bits). At level 1, each peer is connected to its immediate neighbours. At level 2, all the peers whose keys start with 0 are linked and are ordered according to the value of the key; the peers whose key starts with 1 are connected similarly. At the third level, there are 4 lists: one for each combination of the first two bits of the keys. In the basic skip-graph structure each peer stores $2\log n$ pointers (routing tables) to other peers, where $n$ is the number of nodes in the overlay system. The pointers at level $h$ of a node point to nodes at distance $2^h$ of the original node, to the left and to the right.

Search is started at the top most level of the node seeking a key. If required, it continues to lower levels until level 1 is reached. As in skip-lists [12], each peer receiving a data request compares the searched key with the keys stored by neighbour peers at different levels. If the key stored by the neighbour on the highest level is bigger than the searched key, the query is forwarded to the neighbour. If not, the searched key is compared with data stored by a neighbour on a lower level. The comparison process continues until the key is found or the address of the node storing the closest key is returned. Contrary to skip-lists, in skip-graphs a key is searched only in the lists that contain keys starting with the same prefix as the searched key. In skip-graphs resource location and node addition and deletion can be done in logarithmic time.

**Polyhedra for describing a zone**

Polyhedra [13] are mathematical objects used in many fields as in computational geometry. In our case, we use a polyhedron to describe a zone. A zone is a space of possible values for a vector. For example, in a four dimensional space, a zone could be described by all the points in the interval $([100;233[, [100;233[, [100;233[, [100;233])$. One main requirement of our architecture is to be able to test if a point belongs to a zone with low time complexity. Another requirement is to be able to split-up a zone into contiguous sub-zones with low spatial complexity. For example, the previous zone could be divided in two sub-zones as $([100;233[, [100;120[, [100;233[, [100;233])$ and $([100;233[, [120;233[, [100;233[, [100;233])$. In order to perform these kinds of operations we adopt the polyhedral approach. By definition, a polyhedron is a set of the form $\{x \in R^n | Ax \leq b\}$ where $A \in R^{m \times n}$ and $b \in R^m$. A hyperplane is a set $\{x \in R^n | a^T x = b\}$ where $a \in R^n$ and $b \in R$ are given numbers. An example of a polyhedron with two dimensions is:

$$\begin{align*}
(x_1 - 2x_2 &\leq 3) \\
3x_1 - x_2 &\leq 2
\end{align*}$$

We consider that a set of polyhedra is composed of multiple polyhedra of the previous form.

Our interest in using the polyhedral theory is to be able to represent zones using set of inequalities. We use here a space decomposition achieved by hyperplane intersections orthogonal to dimension axis, therefore each zone may be described by a set of polyhedral inequalities. For example in a four dimensional space a zone can be defined by the following conjunction of inequations:

$$(-x \leq -10) \land (x \leq 20) \land (-y \leq -10) \land (y \leq 20) \land (-z \leq -10) \land (z \leq 20) \land (-t \leq -10) \land (t \leq 20)$$

An interesting property of polyhedra is to be able to calculate in constant time cost intersections of polyhedra or simply to check if a point is included in a polyhedron. Our P2P indexing network is built based on polyhedral algebra.

**2.2 The architecture**

The architecture of the routing layer for vectors is identical to a traditional skip-graph network as illustrated in figure 3. It is com-
posed of multiple peers where each peer can store one or multiple vectors. Each peer has \(2 \log n\) pointers to other peers and \(2 \log n\) polyhedra for query routing. At each level of the corresponding skip-list a peer stores several polyhedra that represent the routing criteria. For example in figure, peer \(K\) stores at the first level two zones instead of a numerical value (as in classical skip-graph structures). Thus, \(\text{Left}\_\text{Zone}_K(Z_1)\) determines the zone for the peer \(J\) and \(\text{Right}\_\text{Zone}_K(Z_1)\) determines the zone for peer \(L\). At the second level two other polyhedra characterize the value zones for peers \(I, J\) respectively \(L, M\). The same principle is applied for the other levels.

All zones are contiguous. A zone determines all possible values for one or several neighbouring peers. It is possible for a peer to be located at one extremity of the network and to be truncated and thus the zone to be empty (for example the peer \(A\) on figure has no left neighbour peer). The second layer of network (the indexing layer) is responsible for computing the continuous space of these zones (as presented in the next section).

The remainder of this section concentrates on the adaptation of the skip-graph routing algorithm to our purpose. The problems of constructing a vector-based routing layer are described in the following section.

### 2.3 Vector-based routing

A first difference with traditional skip-graphs resides in the routing table criterion. At each level instead of numerical values we use skip-list zones (described by polyhedra). If a point is included in a zone \(Z_i\) associated to level \(L_i\), then one or several peers assigned to this zone \(Z_i\) and pointed by the level \(L_i\) can solve the searched point.

According to the following property: \(\forall i, j \ i < j \ Zone(L_i) \subset Zone(L_j)\), and based on the remark that in our case the relation order required by the skip-graph routing criteria is translated to the inclusion relation, we have developed the simple routing algorithm

<table>
<thead>
<tr>
<th>Algorithm 1 Point routing algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Input (P, DF); a point (P) to search. A direction (DF{\text{Left, Right, Up}}) fromwhere the request came</td>
</tr>
<tr>
<td>2: Output ; adresse of a Peer</td>
</tr>
<tr>
<td>3: for SkipList (L_i) in level (n) to 1 do</td>
</tr>
<tr>
<td>4: \quad if (P) IN (\text{Left}_\text{Zone}(L_i)) AND (P) NOT IN (\text{Left}_\text{Zone}(L_{i-1})) then</td>
</tr>
<tr>
<td>5: \quad return Adr(Left((L_i)))</td>
</tr>
<tr>
<td>6: \quad end if</td>
</tr>
<tr>
<td>7: end for</td>
</tr>
<tr>
<td>8: for SkipList (L_i) in level (n) to 1 do</td>
</tr>
<tr>
<td>9: \quad if (P) IN (\text{Right}_\text{Zone}(L_i)) AND (P) NOT IN (\text{Right}_\text{Zone}(L_{i-1})) then</td>
</tr>
<tr>
<td>10: \quad return Adr(Right((L_i)))</td>
</tr>
<tr>
<td>11: \quad end if</td>
</tr>
<tr>
<td>12: end for</td>
</tr>
<tr>
<td>13: if (DF) in {\text{Left, UP}} then</td>
</tr>
<tr>
<td>14: \quad forward Adr(ExtremLeft((L_n))) with Left DF</td>
</tr>
<tr>
<td>15: \quad end if</td>
</tr>
<tr>
<td>16: if (DF) in {\text{Right, UP}} then</td>
</tr>
<tr>
<td>17: \quad forward Adr(ExtremRight((L_n))) with Right DF</td>
</tr>
<tr>
<td>18: \quad end if</td>
</tr>
</tbody>
</table>

Contrary to the normal use of a skip-graph which is based on a key made of a sequence of bits, the principle of the routing in our architecture is to use a point \(P\) (e.g. a vector). A peer which must route a point \(P\) will test if the point belongs to one of the polyhedra (zones) of the highest level \(i\) according to the condition (lines 4 to 7 of pre-
vious algorithm): \( P \in \text{Left Zone}(L_i) \) and \( P \notin \text{Left Zone}(L_{i-1}) \). This search is processed for the two directions (lines 4, 7, and lines 8, 12), and the request is sent to the correct peer. If one the searched points is found inside one zone the address of the peer responsible for the zone is returned (as there are no common points for two zones after finding one address the algorithm stops). If no list makes it possible to solve, then the request is sent to the most distant peer received from skip-list \( L_n \). In this case, the forwarded message (lines 14, 17) is propagated according to the received direction of the request (see DF). If the request comes from the peer which initiates the request (DF=up), then the request is sent in both directions. If this is not the case, then the request is simply propagated in the same direction.

Figure 4: Example of the routing process

For exemplification, figure 4 depicts the principle of the routing using zone criteria. We describe the routing procedure at peer P53 for the right part of the skip-graph (the process is symmetric for the opposite direction). Given a point \( P \), peer P53 can evaluate if the point \( P \) is enclosed in a zone interval \( (Z_1, ..., Z_3) \). If \( P \in Z_3 \) it means that a peer from P3 to P9 could store a relevant data item. As the space of values is a tessellation (means that a point is included in one and only one zone), in order to eliminate peers P3 to P6, we can add the second predicate: \( P \notin Z_2 \) when testing at level 3. If the predicate \( P \in Z_3 \land P \notin Z_2 \) is true, it means that the point can be solved by one of the peers defined in the zone E3. Then the request is sent to the peer P9 which is situated at the extremity of the zone Z3. Peer P9 uses the same process but with other skip-lists \( (Z9-x) \) in order to determine which peers, between P2 and P9 can solve the request. On the contrary, if the predicate \( P \in Z_3 \land P \notin Z_2 \) is false, the test restarts with the level -1. If no solution is found in the area from P53 to P9, the process restarts by sending the request to the peer at PE. Then the process restarts.

Yet another example of query routing is presented in figure 5. As already presented each query is routed in the second level named routing layer. To achieve this, each peer tests whether the searched value is on the right or on the left by performing zone inclusion operations. For example, to find the value \( V_1 \) (figure 5) \( P_4 \) tests to see whether the value is inside \( Z_{12} \) or \( Z_{r3} \). As the value is found in \( Z_{12} \) the search process continues on the left part. \( V_1 \) is compared against \( Z_{11} \) and is found to be covered by this zone. The only peer inside \( Z_{11} \) is \( P_3 \) the owner of the key. The process is similar when searching for \( V_2 \). As \( V_2 \) is contained nei-
ther by $Z_{l2}$ nor by $Z_{r3}$ the query is forwarded to the limit of the zones, respectively to peer $P_8$. Peer $P_8$ finds the value included in the first right zone represented by peer $P_9$.

The algorithm for range queries can easily be deduced from the algorithm of routing a single point: the algorithm 2.

<table>
<thead>
<tr>
<th>Algorithm 2: Range routing algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Input PR, DF; PR a zone to search. A direction DFLeft, Right, Up from where the request come from</td>
</tr>
<tr>
<td>2: Output ; address of a Peer</td>
</tr>
<tr>
<td>3: $S = \emptyset$</td>
</tr>
<tr>
<td>4: for skip-list $L_i$ in level n..1 do</td>
</tr>
<tr>
<td>5: if $PR \cap \text{Left Zone}(L_i) \neq \emptyset$ AND $PR \cap \text{Left Zone}(L_{i-1}) = \emptyset$ then</td>
</tr>
<tr>
<td>6: $S \leftarrow S \cup \text{Adr(Left}(L_i))$</td>
</tr>
<tr>
<td>7: end if</td>
</tr>
<tr>
<td>8: end for</td>
</tr>
<tr>
<td>9: for skip-list $L_i$ in level n..1 do</td>
</tr>
<tr>
<td>10: if $PR \cap \text{Right Zone}(L_i) \neq \emptyset$ AND $PR \cap \text{Right Zone}(L_{i-1}) = \emptyset$ then</td>
</tr>
<tr>
<td>11: $S \leftarrow S \cup \text{Adr(Right}(L_i))$</td>
</tr>
<tr>
<td>12: end if</td>
</tr>
<tr>
<td>13: end for</td>
</tr>
<tr>
<td>14: if $S \neq \emptyset$ then</td>
</tr>
<tr>
<td>15: for s in S do</td>
</tr>
<tr>
<td>16: forward to s with PR</td>
</tr>
<tr>
<td>17: end for</td>
</tr>
<tr>
<td>18: else</td>
</tr>
<tr>
<td>19: if DF in {Left, Up} then</td>
</tr>
<tr>
<td>20: forward \text{Adr(Extrem}_\text{Left}(L_n)) with Left DF</td>
</tr>
<tr>
<td>21: end if</td>
</tr>
<tr>
<td>22: if DF in {Right, Up} then</td>
</tr>
<tr>
<td>23: forward \text{Adr(Extrem}_\text{Right}(L_n)) with Right DF</td>
</tr>
<tr>
<td>24: end if</td>
</tr>
<tr>
<td>25: end if</td>
</tr>
<tr>
<td>26: Return S</td>
</tr>
</tbody>
</table>

2.4 Network evolution

Peer log in and log out

A login peer can connect to the skip-graph network as a client searching for data. For that, a peer connects to an already connected peer. The new peer is a shadow/dual of the peer to which it has connected (named master peer). A master peer can have multiple shadow peers. Then, the master peer sends a request to all its neighbours (which are connected by the skip-graph), and tries to determine which peers have the lowest number of shadows. Once this is done, the new peer reconnects to the peer with minimal number of shadows.

The shadow peers assume multiple roles in the architecture. First they can represent a backup of the master peer. Secondly, they can assume part of the incoming requests of the master. But their most important role appears during the splitting process, as explained in the next section, to support the evolution of the indexing layer: shadow peers are used to make new nodes that process new zones.

During the login process, an entering peer can assume a server role in two ways. First as a server of its own data in case it plans on adding data to the network. Secondary, a peer can be a server for the routing process when it is elected by the indexing layer. The logout process is very simple in two cases: first, if a master peer leaves, it transfers his role to a shadow peer. The hub to which it was connected must therefore update its routing data. Secondly, if a shadow peer leaves, no other actions are performed. In any case, if index entries describing local data are previously propagated in the network, other peers must be informed.

Failure Recovery

To conclude this section, we summarize several cases of network evolution in difficult situations as suggested by other systems, that rep-
resent the inspiration point for our network; however, since this is not the main focus of the paper, we will not detail.

- Routing layer failure recovery: In this case, the strategy is similar to traditional skip-graph networks. Different levels of skip-lists are used to repair the broken links. Zone criteria can be calculated using the polyhedral algebra.

- Peer-Failure: This case is identical to peer failure in any other DHT network. Recovery log, mirroring peers etc. are solutions already presented for this problem.

- Peer login and logout: The network mechanism asks an existing peer to replace the functionality of the leaving peer.

3 The indexing layer

In this section we deal with the problem of splitting of a zone of values for the routing layer. We recall that this layer, dealing with space division and data organization is called indexing layer and is part of the general data management system. The goal of this layer is to generate separate zones of data by dividing the domain (space) of possible values. Let us recall that the indexing layer must conserve a property of order translated by the inclusion of zones. Thus, the data-zones of the routing layer must verify:

- \( \forall i, j \quad i \leq j \Rightarrow Zone(L_i) \subset Zone(L_j) \); the zone of higher level must include the zone of lower level.

- if \( P_1 \) has \( P_2 \) as neighbour then \( Zone(P_1) \) adjacent \( Zone(P_2) \). This adjacency propriety assures the continuity of the zones. It is therefore possible to simplify the description of a zone by reducing the set of equations of a polyhedron. Without this property the overall network could still function but each zone would be described by more equations (meaning a high degree of data zone fragmentation).

- \( \forall p, \exists \) zone \( Z \) such as \( p \in Z \)

The hub

The hubs are network components that are responsible for the network evolution and for the description of a set of zones. They represent the basic components of the indexing layer. Besides zone splitting, they can also be used for routing queries (in case of failure of the routing layer).

As illustrated by the figure, each hub has \( N \) children (that are hubs or peers), one father hub (except the root hub as explained later). A hub of size \( N \) has a single input and \( N \)-outputs and is characterized by a set of zones \( \{Z_1, Z_2, ..., Z_n\} \). Each zone is defined by a polyhedron. Each entry value of a hub (e.g. a vector), \( x \) is forwarded to the output for which \( x \) is a solution for the corresponding polyhedron (zone): \( x \) is forwarded to output \( i \) if \( x \) is a solution for \( Z_i \) (figure 6).

By hub-dimension we refer to the maximum number of children that can be connected to a hub. The hub-dimension is a parameter that is considered to be fixed for all hubs in the indexing layer (a strategy with a variable hub-dimension can be also considered). It defines the physical characteristics of a hub.
We define the storage factor for a hub as the sum between the number of connections and the number of polyhedra. For a hub of dimension \( N \) the storage cost is defined as \( N \times (\text{polyhedra} + \text{child\_pointer}) + \text{father\_pointer} \). Compared to other indexing data structures the size of the indexing information stored in a hub is small (several kB in the worst case). It is important to recall that although the size of the hub index is small it has a very important space division role.

Each hub runs on a computer connected to the P2P network elected by a distributed algorithm as described in [17]. Several aspects regarding the persistency and failure recovery are presented in [17]. As the hub performs only space division tasks the overall CPU load is not significant (comparing to usual query execution tasks).

Considering the space division task we present two important properties of a set of zones:

\[
\begin{align*}
Z_1(HUB_k) & \text{ adjacent } Z_2(HUB_k) \text{ adjacent } \ldots \text{adjacent } Z_n(HUB_k) (P_1) \\
\exists j \text{ such as } Z_1 \cup Z_2 \ldots \cup Z_n(HUB_k) & \subset Z_j(HUB_f) \text{ where } \text{Father}(HUB_k) = HUB_f (P_2)
\end{align*}
\]

3.1 The architecture

This sub section describes the dynamic organization and functionality of a set of hubs. In figure 7 we present an instance of the indexing layer. As we can observe the indexing layer follows a tree-like topology where peers represent entities at the lowest levels (leaves) and hubs are situated on the intermediate levels. Zones are divided on the descending paths; the zones from upper levels cover zones from lower levels. For example the zone indexed at the hub \( H0 \) describes all the domain of values. The zone union described by the hubs \( H1, H2, H3, H4 \) represents the zone described by the root hub \( (H0) \) (\( \text{Zone}(H4) = (\text{Zone}(H5) \cup \text{Zone}(H6) \cup \text{Zone}(H7) \cup \text{Zone}(H8)) \)). Recursively each hub describes a sub-zone of its father’s zone. By using a tree topology, zones are split continuously. In the same time, each peer can store any value described by the first upper level hub.

3.2 Evolution of the indexing layer

The dynamicity of the indexing layer is due to the peers that often change their state: their load depends on the number of values indexed at a given moment. As the peers represent the entities at the lower levels, the layer is divided in two parts: the upper part, stable most of the time and the lower part with a more dynamic behaviour. Depending on their data load, peers may divide or prune.

Peer division: When a peer \( P \) is saturated, it must split its load to other peers. To do this, an important problem is to find a set of inequations that divide the set of \( n \) points in two or several sets of points with approximately equal length (e.g for dividing in two we may want to generate sets of \( n/2 \) points). Splitting in two sub sets is not an easy task. One simple solution is to iterate through the set of vectors for each dimension until a set of good splitting equations is found. However the actual polyhedron generation algorithm is
much more complex as intermediate solutions must be reconsidered.

As an example consider a network indexing vectors of 3 dimensions \((x_1,x_2,x_3)\) and a peer \(\Pi\) indexing the set of 4 values: \(\{v_1,v_2,v_3,v_4\} = \{(0,0,0),(0,0,2),(0,2,0),(2,0,0)\}\). We want to evenly distribute the load of the peer to other two peers. If we take into consideration the values of the first dimension we observe that we can make a split if we decompose the values in a set that respect \(x_1 = 2\). In this case the decomposition generates a set of 3 values and a set of one value. As it can be observed the values are not evenly distributed and a different decomposition must be found. Similarly we can find decompositions on the second and on the third dimensions with the same (uneven) distribution of values. In order to find a good solution, at least two dimensions must be considered. In Algorithm 3 we present the general algorithm for polyhedron generation.

The algorithm splits a peer data set \(S\) into two subsets that are evenly distributed. If the data quantity \(qt\) associated with the two sub-sets is not equivalent the process restarts and tries another dimension (next SA in line 9). This is the first version of the algorithm; another version of the splitting algorithm that takes into consideration multiple axis when doing the splitting can also be considered.

We illustrate the zone decomposition by the following example. Consider a zone \(zone(S) = ([100,233],[100,233],[100,233])\) and consider that the splitting is done on the second axis (SA=1). In this case generated results are \(zone(S_1) = ([100,233],[100,x],[100,233])\) and \(zone(S_2) = ([100,233],[x,233],[100,233])\). It is simple to generate new polyhedra based on properties P1 and P2, defined in section 3.

Whenever a peer divides its load to other peers the upper level hub must evolve in order to adapt to the new load. The hub evolution is similar to the B-Tree division algorithms. For an entry \(h_i\) of a Hub that is connected to \(P\) (noted \(h_i(P)\)), the algorithm performs as in Algorithm 4.

To clarify this algorithm: if the dimension of the hub allows an additional entry (lines 4), the old entry is replaced (lines 5 and 6) by two new entries. The new entries refer to the two peers \(P_1, P_2\) with their associated sets \((S_1, S_2)\) and their associated zones. These two sets are previously computed by the algorithm. If the hub reaches its maximum dimension, a new hub is created (line 8). The current hub node replaces the old entry with the new hub (line 8). The new hub node points to the splitting peers \(P_1, P_2\) (lines 10,11). In this way the layer tree grows at its basis with one more level. Note that during the
Algorithm 4 Hub splitting algorithm

1: Input : hi(P), P1(S1, zone(S1), P2(S2, Zone(S2)), S content of the splitting peer
2: Output : Modified state of the network
3: Hub.Remove Hi(P)
4: if Dimension(Hub) ≤ N − Dimension
5: Hub.Append(E(P1, zone(P1)))
6: Hub.Append(E(P2, zone(P2)))
7: else
8: Hub_New is a new Hub
9: Hub.Append(E(New_Hub, zone(S)))
10: Hub_New.Append(E(P1, zone(S1)))
11: Hub_New.Append(E(P2, zone(P2)))
12: end if

execution of a hub split, only a maximum of four nodes are involved.

Load Balancing: Each peer divides uniformly its load in such a way that the load of each new peer roughly reaches 50%. As the measurements prove, the average load of all peers is over 50%. This means that each peer plays an important part in the total load and there are no idle peers. The Hub-Network overlay tree may not be balanced (overall same height). However, as we will show, this does not present any impact on the query routing performance.

Removing indexing point: When removing a peer the indexing layer reacts similar to B-Tree pruning techniques. As the zones from upper levels cover the zones from the lower levels no major network transformations are required.

Robustness and failure Technical solutions regarding the tree network evolution in case of failure can be found in [17], while a detailed robustness analysis is presented in [16].

Indexing values For point indexing, each peer that is connected to the routing layer sends a request for routing a point to the right peer. At the right peer, the value is stored if the capacity allows. Otherwise, the peer division process is triggered. Note that this routing uses mainly the routing layer while the indexing layer is not solicited.

4 The Overall P2P Indexing Architecture

Based on the previous descriptions of routing layer and indexing layer we present the overall network architecture used for publishing and querying multidimensional values as vectors. We recall that the routing layer is used for finding a peer that indexes a certain value based on a query expressed by a vector. The proposed architecture efficiently routes multidimensional queries based on the polyhedral comparing criteria and on the skip-graph specific query routing algorithm.

Our problem is to compute the polyhedra criteria for each level of a skip-list according to the skip-graph principle. Remember that at each level, we have two pointers: one to the left part and one to right part. According to the basic description of skip-graph, for the first level, the list references neighbours located at a distance of 2. At the second level, it references a peer at a distance $N^2$, at the third level, $N^3$, etc.

For this algorithm we will use the routing

![Figure 8: Overkay network](image-url)
layer in order to obtain the pre-computed polyhedron for a zone. According to the properties described in section 3, in figure we remark that at the level 1 of the routing layer, given a peer (P53), its hub (H41) defines the Zone(H41) that controls N neighbours peers (N is the dimension of the Hub, and in the figure its value is 7). At level 2 of the routing layer given one peer (P53), its hub H33 controls N peers. In this case, Zone(H25) describes a zone with potentially $N^2$ peers. Based on these remarks, we can get directly the polyhedron from the right hub in the routing layer. For example, for the peer P53 of the figure at level 1, we can get from H41, the left part, represented by Zone(P1) $\cup$ Zone(P2). From H41, we can also get Zone(P3) $\cup$ Zone(P6). For level 2, peer P53 can get all zones from H35 to H41. The process is identical for the right part. Let us stress that two levels are enough in order to cover the whole local domain of a peer. We also make the observation that figure only presents a single skip-list at a given level, if we analyse the situation from the skip-graph point of view.

In case of failure of the indexing layer the zones can be recomputed by gathering information from the neighbouring peers. Recall that each node can be reached even in the case of a skip-graph failure. After the nodes contact each other, the zone generation is trivial (but still requires the exchange of several messages).

**Emergency routing** Although the indexing layer is mainly used only for organizing the network content (the indexed vectors), it can be also used for routing in case of emergency (when the routing layer is down).

The routing principle is described as follows: according to figure if a peer P connected to the Hub H5 needs to send a request R thought the network, this must be compared against all zones defined by H5. If the searched point is included in one of the zones defined by the hub, the request is forwarded downwards to the corresponding child. On the contrary if the request does not verify any of the zones it is simply sent to the father of the hub and the process repeats. Note that, statistically speaking not all the requests reach the root of the indexing layer.

5 Experimental Evaluation

5.1 Experimental Setup

For comparing our system with similar P2P systems we generated an application specific data set and we stressed in a similar manner our implementations of different systems. This section aims to validate by measurement two main features of our architecture:

- the scalability of our approach in terms of indexing and routing regarding the size of a vector (e.g. number of dimensions of a multidimensional point).
- the scalability in terms of number of hops required when routing a query.

All algorithms presented in this paper were implemented on a PC equipped with 1Go of memory. As all peers run on only one machine, therefore in this setup no time measurements can be given (however, these remain less important in our evaluation as we intend to present the global network behaviour). To estimate the costs of the algorithms, measurements are given according to quantifiable values such the numbers of hops, and the number of peers. These measurements make it possible to deduce experimental algorithm complexities. The data sets are generated randomly based on some parameters that control the point distribution so as to concentrate a more or less significant number of points in a
given area. These data sets aim at showing the contribution of our work and the limits of several existing architectures (i.e. Chord). The results show that in most of the cases our architecture has better performances.

5.2 Experimental Evaluation

Peer load scalability. We measure how the load of each peer varies with the size of the vector to be indexed. We show that even for lengthy vectors the overall peer load is uniformly distributed. The test scenario consists in indexing 1000 points in our network and in Chord/Z-Order network and measuring the load for each peer. To increase the difficulty of the problem that we try to solve, points are generated in a N dimensional zone where N varies from 4 to 512. Each peer has a fixed loading capacity of 128 points for emphasising the network evolution. The results are presented in figure 9. We notice that the minimum and maximum load rates for peers are stable at 48.56% respectively 99.88%. The results show that there are no idle peers (peers not indexing values) as each peer is loaded by at least 48%. We can also observe that the vector dimension does not have a great influence on peer load.

Network characteristics. In figure 10 we measure the number of peers required by the network in order to adapt to various number of points to be indexed. The dimension of each hub is 8, meaning it may have a maximum of eight children. We can observe that the required number of peers is roughly proportional to the number of points with an average load of 75% for each peer. Other measurements show that by increasing the dimension of a hub for the same number of peers we can significantly reduce the total number of hubs. Similarly in figure 11 are presented the required number of hubs for adapting to an arbitrary number of peers.

![Figure 9: Peer Load Rate](image9)

![Figure 10: Data load analysis](image10)

![Figure 11: Hub scalability](image11)
Average Number of Hops. In this test scenario each peer can randomly decide to search for a point located on another peer. We consider that each peer can load 1024 points. We trace on the network the number of exchanged messages. We try to best simulate a real situation by constructing the network and data set progressively by adding peers and vectors incrementally. For each network size we measure the number of hops generated by 1000 data requests and compute the average number of hops as the ratio between the total number of hops and the number of queries. During the measurements we insert vectors of a variable size (8, 16, 32). As observed in figure 12 we have a logarithmic evolution that depends on the number of nodes. In figure 13 we present a comparative image of a mathematical evaluation of Chord.

![Figure 12: Number of hops required for solving a query](image)

Figure 12: Number of hops required for solving a query

Number of nodes involved in network evolution. Figure 14 measures the ratio between the number of indexed points and number of nodes (peers and hubs) involved in the indexing process (the number of nodes required for routing a point to the good peer is not considered). The number of peers required for routing is given in the previous sub-section, as it is twice larger than the result presented in figure 14; we do not include it here because it masks the current observation. However, the curve includes the interactions with all hubs that are connected to an updated peer, in order to update its skip-lists after a splitting. As an example we observe that to index 81000 points 81598 nodes are involved, the ratio being 1.00738. Each peer can load only 1024 points (voluntarily small), the dimension of a hub is 8, and the dimension of vectors is 16. This curve shows that the indexing process has a very low cost despite of the updates of the skip-lists and the indexing layer: our method is scalable. The increase in the curve is due to the part of the indexing layer which grows bigger, but also to the number of levels of the skip-lists that becomes larger and larger.

Number of messages generated by a peer split. When a peer is overloaded and it must split its load we distinguish two evolution cases: (i) If the parent hub can index another peer then the total number of exchanged messages is composed of: the number of messages of finding a new peer (zero if a shadow peer exists, one if there is no local shadow peer), number of messages for sending half of the actual content (one message) to the new peer, number of messages for sending new zones to the parent hub (one message). (ii) If the par-
ent hub can not index another peer another
message must be exchanged for finding a new
hub. From this analysis we observe that the
total number of messages exchanged on a peer
split is constant and does not depend on the
number of peers indexed into the network.

6 Related Work

The main problem addressed in this paper
is the proposition of an architecture suitable
for indexing complex data with many dimen-
sions (multidimensional data) that offers the
possibility of executing queries other than
simple selections (ie. range queries or wild
card queries). Most database systems offer the
applicative support for a powerful querying
language. Recent works try to introduce
the same support in the world of distributed
and P2P systems. In this section we try to
present a comparative analysis of existing P2P
indexing techniques regarding their querying
support.

Most of the multidimensional indexing solu-
tions are based on the idea of converting of a
multidimensional domain of values to a spe-
cific domain that can be exploited by existing
P2P technologies such as DHT systems ([3],
[4], [7]). These solutions have the advantage
of having a good background in Geographical
Information Systems. But most of these
domain transformation techniques are not
scalable as a result of the limited size of the
routing indexes (SHA [9] uses 160 bites). For
data with a lot of dimensions, a representation
limit is imposed. Moreover for data domains
where most of the values are concentrated in
several sub-domains the domain translation
techniques generate a bad value distribution
followed by a bad load balancing (in the P2P
context).

Other methods have also been proposed that
are not based on domain transformations
([2], [11]). In these cases each dimension of a
multidimensional value is indexed using an
indexing system suitable for indexing simple
values. These architectures are very well
adapted for indexing values with many dimen-
sions but the overall structure is much more
complex (because of the many interconnected
structures) and the number of hops required
for routing a query is significant.

Our architecture is similar in some aspect to
the first approach since we use a structure
inspired by skip-graphs and to the second
approach since we directly index values
composed of multiple dimensions. On the
contrary we do not use any domain translation
techniques but we use the polyhedral algebra
to describe domains of multidimensional
points. The use of polyhedra gives us the
possibility of describing the domain indexed
by each peer with an "infinite" level of detail.
In the same time our method provides good
load balancing no matter what properties the
data may possess.

The existing works [4] [10] [3] [1] [2] give
solutions for range query routing with per-
formances between \( O(\sqrt{n}) \) and \( O(\log n) \).
The routing is based on interpreting the
new domain coding (for the first category of
systems). From the second category, Mercury
[11] also proposed "wild card" routing (as in
One important observation is that all systems proposing a good load balancing domain mapping method do not offer good support for range query execution (e.g. Chord is not adapted to range query execution because of its mapping/hashing method). Other systems offer good range query support by splitting the domain of values based on a domain splitting function shared by all peers. However in this case load balancing problems appear due to non-uniform distribution of values in the indexed domain. To enforce load balancing, several solutions were explored: load negotiation between an overloaded peer and idle peers, increasing the number of peers in an overloaded zone [10], dynamic domain division [3]. The dynamic domain division solution assures a better peer loading but its weakness is that all the peers must know at any moment the current domain division "function". Consequently at given moments the P2P network is in inconsistent states: the actual domain division function is not shared by all peers. Contrary to this approach, we propose a solution of dynamic domain division based on a meta-P2PNetwork that does not require any sharing of global information between all the peers.

7 Conclusion

In this paper, we have presented a P2P architecture suitable for multidimensional value indexing. The scope of our work is the introduction of a scalable architecture that can be used to index lengthy vectors. To this end we propose a P2P network composed of two independent layers, one used for routing and the other used for data distribution. Based on the remarkable properties of skip-graphs and on the space division characteristics of a tree-like network we offer a solid, fault tolerant, architecture that respects balanced load distribution. One main problem we try to solve is to offer a P2P indexing model that is not sensitive to the complexity of the values to be indexed, that does not destroy the data properties (i.e. order of elements) and that respects the P2P paradigm. In order to achieve this goal we have introduced the polyhedral theory to the P2P domain and adopted a network model that can be used for splitting a large value domain in continuous zones distributed to the participant peers. In the future we plan to develop specific database applications (i.e. XML indexing) using the network model proposed in this paper. Nevertheless we plan to offer more measurements that show the strengths of our architecture compared to other similar systems.

References


[16] H. V. Jagadish, Beng Chin Ooi, Quang Hieu Vu: BATON: A Balanced Tree Structure for Peer-to-Peer Networks. VLDB 2005: 661-672