Enhanced Lookup Queries for Large-Scale Distributed Data Structures

Jordi Pujol-Ahulló and Pedro García-López

Universitat Rovira i Virgili, Department on Computer Science Engineering and Mathematics, Tarragona, Spain

Abstract. Structured peer-to-peer systems are a class of distributed data structures that provided initially only exact-match queries in a scalable way. Exact-match queries consist of retrieving single values from the network (i.e., put(key, value) / value ← get(key)), in such a way that if the data exists, it is returned. Unfortunately, this service is not enough for modern distributed and parallel applications. Thereafter richer queries have been the focus on the field of distributed systems based on peer-to-peer networks. Range queries, K-nearest neighbour queries and spatial queries belong to the family of similarity queries. This sort of queries are used by a wide range of applications. This chapter performs a comparison of the last and most remarkable works in this field. To do so, a set of evaluation parameters are defined with which all systems are compared to. Examples of the analysed parameters are: overlay network topology, application domain, query correctness, completeness, storage and time efficiency and load balancing, to say the least. The goal, thus, is to provide an insight on peer-to-peer-enabled distributed and parallel algorithms that support similarity queries in the large scale.

Keywords. Parallel algorithms, Similarity queries, Range queries, K-Nearest Neighbour, Spatial queries, Peer-to-Peer networks

Introduction

Unstructured Peer-to-Peer Systems (UPS) appeared earlier to bring together edge resources. File sharing applications are predominant in this context, where Gnutella embodies UPS. They are called unstructured because links between nodes are established arbitrarily. Nevertheless, UPS suffers of two main drawbacks: lack of query correctness and overhead on communication. The former can be explained as follows: suppose there is a file in the UPS and a user wants to retrieve it; if such a file is too far from the user, it might not to be found by the user query. The latter occurs because communication in UPS is performed mainly by flooding and causes a high amount of signaling traffic in the network. Hence such networks typically have very poor search efficiency. Structured Peer-to-Peer Systems (SPS) appeared to overcome the problems appeared in UPS. SPS are to provide correctness, a key property for modern applications, as well as routing and time efficiency. SPS are also broadly called distributed hash tables (DHTs) because most of them associate the data owner node by means of a consistent hashing, in an

1Corresponding Author: Jordi Pujol Ahulló, Universitat Rovira i Virgili, Department of Computer Science Engineering and Mathematics, Av. Països Catalans 26, 43007 Tarragona, Spain; E-mail: jordi.pujol@urv.cat.
analogous way to that of traditional hash table’s assignment of keys to a bucket. SPS provide exact-match queries with correctness and applications can use the API `put(key, value)/value←get(key)` to access to the content.

Unfortunately, exact-match queries are not enough for modern applications. Similarity queries consist of a family of queries that provide high-level abstractions to applications, such as range queries, k-nearest neighbours and spatial queries. These kinds of searches enable the user to retrieve data objects related or similar to the query. But SPS did not deal with these kinds of searches initially. Instead, we would need to flood the entire network for retrieving all related content, which becomes unacceptable for large-scale networks. This is why similarity queries have been receiving a lot of attention since the SPS existence, because peer-to-peer qualities can greatly help to nowadays applications. Nevertheless, their different topological approaches and intrinsically parallel algorithms exhibit various trade-offs and operational costs, what makes them different and suitable only for certain scenarios. Thus, we analyse in this chapter some of the last and most remarkable solutions in the field of similarity queries on P2P-enabled large scale parallel systems through a common evaluation framework.

**Guidance by Examples**

To let the reader understand the necessity and complexity of providing similarity queries over distributed and parallel systems based on SPS, we explain two applications where such a functionality is required, among scalability by service decentralization and high performance by query parallelization. Current real-life applications cannot be understood without such richer abstractions.

The first application we consider is the **document retrieval** in a distributed network of workstations (namely nodes). Documents consist of any kind of text documents and the goal of the application is to provide similar documents according to the **user interests**. User interests are specified by means of keywords, like “Mathematics” or “Computer Science”. Actually, each search is an instance of a **k-nearest neighbour query** (k-NN): the system retrieves the **k most similar documents** to the user interests (See Fig. 1(b) for an example). The key procedure of this kind of applications is to treat each document as a pair of the form `{<keyword-list>, <document>}`. The **keyword-list** is a list of keywords that contains the most representative descriptive terms from the **document**. This technique is called **Information Retrieval (IR)**, for what several ways of parallelizing document indexation and search have been widely studied [1]. Particularly, we focus on the distributed document retrieval, so that different networked nodes must be asked for retrieving related documents to the user query.

The second application we consider is a distributed **geographical service location**, so that system participants are networked nodes that possess partial information and help on query resolution. For simplicity and for the proposal of this chapter, we consider the goal of this application is telling the **services location** related to the **user interests** that appear in an arbitrary **geographical area**. This exemplifies the behavior of a **range query**: retrieve all elements from the given range, that is, the geographical area. In addition, this is also an example of **spatial query**, because the goal of the application is locating information from a geographical region (See Fig. 1(a) for an example). This case, **user interests** specify both the geographical area and the sort of services she is looking for. For instance, reader can be interested in retrieving the “Italian Restaurants” locations around
her city. The idea of this application is to retrieve the whole set of items that appear in a rectangular area of arbitrary size. This is however a hot topic on the study of Geographical Information Systems, where the feasibility of search parallelization has largely been studied for different environments (for instance for Grid computing [2]). Clearly, parallelized queries accelerate search operations, reducing the response time, but at the cost of a higher use of communication bandwidth.

All these (distributed) applications present the same elements in their architecture: a **multi-dimensional data domain** with data objects being stored and searched; a **distance function** which defines the similarity between two data objects; and the **algorithms** for storing and searching the content. For instance, IR applications characterize each text document by a $d$-dimensional vector with the best descriptive terms. Geographical service location applications consider each service generically as being a $d$-dimensional vector, including the latitude, longitude and meta-data from the service. In both cases, a search is processed by evaluating the similarity of various data objects according to some distance function, e.g. Cosine distance for IR ones and Euclidean distance for service location ones. Out of these 3 elements, we include into the study both the multi-dimensional data domain and the algorithms. Distance functions are not considered because they are tightly related to the application context, with no effect on the distributed system.

The rest of the chapter is structured as follows. We define formally all similarity queries considered in this study in Section 1. In Section 2 we describe where parallel computing is performed within SPS. In Section 3 we analyse the parameters we include in the evaluation framework. Having all the evaluation tools, we then perform the comparison study of systems providing range queries (Section 4), k-nearest neighbours (Section 5) and spatial queries (Section 6). This chapter closes with the description of some open issues in Section 7 and the concluding remarks in Section 8.

### 1. Enhanced Lookup Queries

The services we discuss in this chapter belong to the family of similarity queries. In particular, we will analyse different systems that provide range queries, k-NN queries...
<table>
<thead>
<tr>
<th>Query Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similarity queries</td>
<td>Similarity queries consist generically on retrieving similar information to the given query. Let the operation $\varphi : Q \times O \to D$, be the distance measure from data items to the given query, where $Q$ is the query domain, $O$ the data items domain, and $D$ the distance domain. Thus, the result of a given similarity query $q$ is formed by the following set: $A = { o \in O</td>
</tr>
<tr>
<td>Range queries</td>
<td>Also known as window queries, this sort of queries provides the whole set of elements into the specified range. There are two approximations to it. Let us call them as sphere-based and region-based approaches. In a sphere-based approach, the range query is defined as a pair $q = {p, r}$, where $p$ is a point in the data domain, $r$ is called the radius. The result set is formed by: $A = { o \in O</td>
</tr>
<tr>
<td>K-NN queries</td>
<td>A given k-NN query asks for the set of elements $A \subseteq O$ such that $</td>
</tr>
<tr>
<td>Spatial queries</td>
<td>Depending on the application context, spatial queries can be considered either a subset of range or k-NN queries, where one of the dimensions to be considered is the location of the data objects.</td>
</tr>
</tbody>
</table>

Table 1. Formal definitions of evaluated query types.

Broadly speaking, similarity queries family provides a set of abstractions to (distributed) applications in order to discover alike resources to user interests. In particular, and recovering one of our examples, IR applications embody k-NN query resolver systems: the application retrieves an arbitrary amount of similar documents to the user-selected keywords. The number of elements in the result set is defined to be as maximum as a system-wide or user-modifiable property value. The system calculates the distances from the user-selected keywords to the $d$-dimensional descriptive terms of documents and builds a result set with the $k$ closest documents. The idea is similar to have all documents ordered in a list from closer to farther to the user-selected words and selecting the first $k$ documents from the list. A centralized solution takes advantage of containing locally all information, possessing a global knowledge of the dataset, and then employing efficient IR techniques to produce the result set. The problem appears when the system is widely distributed among nodes for scalability. The way that documents and information are distributed through nodes will drastically determine the system’s efficiency and response time.
2. Parallel Computing on Similarity Queries

Similarity searches onto SPS benefit from scalability by service decentralization and high performance by query parallelization. Since the service decentralization is clear because they are set up onto a SPS, we delve into query parallelization in this section. Parallel computing of queries provides high performance to the system and this depends mainly on the way nodes are interconnected. For an analysis on the factors that enable parallel computing in a distributed setting refer to Section 3.3.

The way that nodes are organized into the SPS is known as topology. Topology influences the communication costs between participants and the way that information is stored into nodes for global efficiency. The most common peer organizations among SPS are flat topologies (e.g., rings [3,4], torus [5]) and hierarchical topologies (e.g., trees [6]). Broadly speaking, the topology defines a graph $G(V, E)$, where vertexes $V$ are nodes, and edges $E$ the links established between nodes. In addition, only edges $E$ are used for node inter-communication in normal SPS operations.

As we have detailed before, similarity queries considered into this chapter are broadly of two types, so that we describe two generic algorithms, one for range and another one for k-NN queries. Remember that, according to the system’s approach, spatial queries can be a sub-set of any of them. In the worst case, queries are initiated by nodes that are not responsible for resolving the query. This way, these algorithms broadly describe a two-phase problem solver. The first phase consists of routing the query until reaching nodes responsible to answer the query. Afterwards, the algorithm collects all data objects that become the result set within the second phase. These algorithms provide a uniform and generic approach to solve the queries. Thus, the reader will realize that systems considered into this study employ adapted algorithms to suit the particular systems’ architectures.

2.1. Range query parallelization

Algorithm 1 range_query gives the details of how range queries are resolved. Their resolution is fully parallelizable (lines 3–6), so that nodes provide to the querying node their (partial) result sets (line 8). This algorithm employs several local functions. Function subqueries(node) returns the (keyspace) segments that node is responsible for. This way, $lquery$ (line 1) takes a part of the $query$ (namely sub-query) that node can answer, what leaves in $oquery$ (line 2) the part from the $query$ that must be resolved by other nodes. Function $subquery \in oquery$ (line 3) provides all different subqueries from $oquery$, where every subquery is a sub-query that local node is able to start in parallel. Whenever $oquery = \emptyset$, this function provides no subquery so that the loop is never executed. Function best_neighbour(node, segment) returns the best node’s neighbour from node’s routing table to forward the segment sub-query. This selection is SPS-specific and is dictated by system’s routing scheme. Function data(node) provides the data that node possesses (line 7). Once a node has a (partial) result_set, it sends back to the querying node qnode by means of the function send(node, qnode, result_set) (line 9).

As it is explained before, (sub-)queries are only initially routed (lines 3–6). Once responsible nodes are reached, lines 3–6 distributes the (sub-)queries until the whole query is processed ($query = \emptyset$). Most of hierarchical and some of the flat topologies pro-
Algorithm 1. range_query
Input: node /* node where algorithm is executed */
Input: query /* range query */
Input: qnode /* querying node */
1: lquery ← query/subqueries(node)
2: oquery ← query/lquery
3: for all subquery ∈ oquery do {in parallel}
4: neigh ← best_neighbour(node, subquery)
5: range_query(neigh, subquery, qnode)
6: end for
7: result_set ← data(node) ∩ lquery
8: if result_set ≠ ∅ then
9: send(node, qnode, result_set)
10: end if

vide disjoint paths (sequences of nodes) to reach other nodes, what makes Algorithm 1 very effective. Thus, in this scenario, this algorithm parallelizes both routing and query resolution. But most of the flat topologies usually provide path convergence, what means that paths to a common destination intersect at a rate proportional to the distance between the source nodes. In addition, most of the flat topologies providing such enhanced lookup algorithms preserve data locality, so that neighbour nodes in the network own close data objects. In practice, path convergence and data locality imply that the different (sub-)queries sent by the querying node travel the network along the same path. Therefore, these messages waste bandwidth and computer resources until the first responsible node of the query is reached. This is why topologies with path convergence and preserving data locality improve the range_query algorithm (see Algorithm 2) by avoiding query splitting whenever query is only routed (lquery = ∅ in line 2).

Algorithm 2. range_query with path convergence
Input: node /* node where algorithm is executed */
Input: query /* range query */
Input: qnode /* querying node */
1: lquery ← query/subqueries(node)
2: if lquery = ∅ then
3: neigh ← best_neighbour(node, query)
4: range_query(neigh, query, qnode)
5: else
6: oquery ← query/lquery
7: for all subquery ∈ oquery do {in parallel}
8: neigh ← best_neighbour(node, subquery)
9: range_query(neigh, segment, qnode)
10: end for
11: result_set ← data(node) ∩ lquery
12: if result_set ≠ ∅ then
13: send(node, qnode, result_set)
14: end if
15: end if
2.2. k-NN query parallelization

Algorithm 3 provides a big picture on the way k-NN queries are worked out. For the sake of simplicity, this algorithm is based on a naive linear scan, even though other approaches (e.g., space partitioning, locality sensitive hashing and other approximate nearest neighbour search) are actually used by systems in order to improve k-NN query performance.

The \textit{knn} \_query algorithm is as follows. Firstly, a querying node \textit{qnode} performs the k-NN query by specifying a pair \( q = \{ p, r \} \) and \( k \), where \( p \) is a point or data object, \( r \) defines an initial radius and \( k \) the number of data objects to fetch. This operation initially consists of a single message routing (lines 1-3), which cannot be parallelized. When the corresponding node responsible of \( p \) is reached (line 4), it initiates an \textit{expanding ring} search (lines 5-18). This search describes a linear scan through close nodes to corresponding node, in order to provide at most \( k \) objects that are not farther than \( r \) from \( p \). This algorithm though takes the assumption that closer nodes will always provide closer data objects to \( p \) than farther nodes. Thus, the linear scan is consecutively \textit{expanded} in parallel to farther nodes from corresponding node, since the k-NN query is not resolved (line 9). When no new data objects are appended to the candidate result set (having \textit{result} \_set = \textit{new} \_result in line 9), the expanding ring search stops and the k-NN query concludes (line 19).

\begin{algorithm}
\caption{knn\_query}
\begin{algorithmic}[1]
\State \textbf{Input:} node /* node where algorithm is executed */
\State \textbf{Input:} p /* point of the k-NN query */
\State \textbf{Input:} k /* number of elements for the k-NN query*/
\State \textbf{Input:} qnode /* querying node */
\State 1: if node is not responsible for \( p \) then
\State 2: neigh \leftarrow \textit{best\_neighbour}(node, p)
\State 3: knn\_query(neigh, p, k, qnode)
\State 4: else
\State 5: visited\_nodes \leftarrow \emptyset
\State 6: result\_set \leftarrow \emptyset
\State 7: new\_result \leftarrow \textit{knn}(p, k, \infty, \text{data}(node))
\State 8: \( r \leftarrow \textit{longest\_distance}(p, \text{new\_result}) \)
\State 9: while result\_set \text{\#} new\_result do
\State 10: result\_set \leftarrow new\_result
\State 11: for all \textit{nnode} \in \textit{expanding\_ring}(node, visited\_nodes) \text{ do} \{in parallel\}
\State 12: ask \textit{nnode} for partial\_result \leftarrow \textit{knn}(p, k, r, \text{data}(\textit{nnode}))
\State 13: new\_result \leftarrow new\_result \bigcup\text{partial\_result}
\State 14: visited\_nodes \leftarrow visited\_nodes \bigcup\{\textit{nnode}\}
\State 15: end for
\State 16: new\_result \leftarrow \textit{knn}(p, k, r, new\_result)
\State 17: \( r \leftarrow \textit{longest\_distance}(p, \text{new\_result}) \)
\State 18: end while
\State 19: send(node, qnode, result\_set)
\State 20: end if
\end{algorithmic}
\end{algorithm}
As before, this algorithm employs some node’s local functions. The condition ‘node is not responsible for p’ is only false whenever node owns p. This means that if p existed, node would be the node to hold p. Function best_neighbour(node, p) (line 2) is the same as before, but where the neighbour is selected according to the searching point p instead. Function knn(p, k, r, \{dataset\}) (lines 7, 12, 16) provides at most k data objects from dataset that are not farther than r from p. At line 7, r has not been setup, so that a worst case \( \infty \) value is given. Function expanding_ring(node, visited_nodes) (line 11) returns a list of nodes that should be visited in the next query’s round as the expanding ring search dictates. Note that visited_nodes holds the set of nodes close to node that the expanding ring scan has already visited. Line 12 tells node to wait for the results partial_result from neighbour node nnode. Thus, an inter-node communication is performed here between node and nnode. Function send(node, qnode, result_set) (line 19) is the same as before, even though this function is only executed once in knn_query algorithm.

These algorithms provide a global view on how similarity queries considered into this study work. We provide in the following section the significant parameters with which we will evaluate all systems considered into this study.

3. Evaluation Criteria

The systems being studied along this chapter are analysed accordant with a set of qualitative and quantitative parameters, constituting a common evaluation framework (see Table 2). For a fair comparison, it is highly recommendable to consider both quantitative and qualitative parameters that describe the key systems’ components. In addition, such an evaluation framework provide the systems’ portrayal and facilitates the comparison between systems. The considered parameters refer to structural, efficiency, as well as behavioral properties of systems. The parameters can be considered broadly of two types: a) implementation parameters, that refer to the system design and construction, and b) quality of service parameters, that consider the quality of the query resolution. An analysis of parallel computing abilities according to these properties is also included.

3.1. Implementation Criteria

Which parameters do we consider for their inclusion into the evaluation framework? Let us consider the geographical service location application to this end, distributed over the world onto an arbitrary number of nodes. As we have seen before, the way that nodes are organized (namely the topology) dictates the communication costs between nodes and the way that information is stored into nodes for global efficiency. This motivates the inclusion of the topology into the evaluation framework.

We know how nodes are interconnected, but how are nodes addressed into the system? Every SPS specifies an identifier space (namely keyspace), and nodes are provided with a single identifier from the keyspace. By applying a SPS-specific routing algorithm, messages targeting a node I reaches such a node (namely owner node), no matter which is the sender. For instance, in Chord [3] nodes are identified by a single value in the natural range of numbers of \([0, 2^m)\), where m takes a default value of 160. This keyspace is said 1-dimensional or unidimensional and is the most common within SPS.
<table>
<thead>
<tr>
<th>Criteria</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Topology</strong></td>
<td>The term <strong>topology</strong> refers to the way that nodes are organized into the distributed network. The topology can be broadly specified as <strong>structured</strong>, when the topology relies on any geometric form (e.g., ring, hypercube, tree, etc.), and <strong>unstructured</strong>, when nodes are connected with each others but without a clear structure (i.e., mesh).</td>
</tr>
<tr>
<td><strong>Dimensionality</strong></td>
<td>Systems that provide this kind of high-level services in a distributed way, can support either one-dimensional (<strong>1-D</strong>) or multi-dimensional (<strong>M-D</strong>) application domains. When possible, systems will be analysed for both sorts of dimensionalities.</td>
</tr>
<tr>
<td><strong>Mapping approach</strong></td>
<td>Because in most cases application domains do not correspond to SPS keyspaces, the distributed application needs to perform a transformation to the information in order to allow its indexation. This kind of transformation is called also <strong>mapping</strong> and defines exactly the way the transformation is achieved. We can find these sorts of transformation:</td>
</tr>
<tr>
<td></td>
<td><strong>1:1</strong> (Left) One application dimension is mapped to a single SPS keyspace (right).</td>
</tr>
<tr>
<td></td>
<td><strong>M:1</strong> The whole multi-dimensional application domain is mapped to a value in the SPS keyspace.</td>
</tr>
<tr>
<td></td>
<td><strong>M:P</strong> The whole multi-dimensional application domain is transformed to a P-dimensional SPS keyspace.</td>
</tr>
<tr>
<td></td>
<td><strong>N/M</strong> When no mapping is applied.</td>
</tr>
<tr>
<td><strong>Storage efficiency</strong></td>
<td>In a distributed or parallel system, the amount of storage used to record information is significant to reduce notably the response time. In general, the more copies of the same information, the less delay on answers. We use this property to quantify the <strong>overhead</strong> that different algorithms pose on storage.</td>
</tr>
<tr>
<td><strong>Time efficiency</strong></td>
<td>This property measures the amount of time needed by the distributed algorithm to return the whole set of results. Because in most of the algorithms nodes with requested information answer directly to querying node, this time is considered to be equivalent to the amount of time the algorithm takes for visiting the last node involved in the query. When noted, it also means the time that a first (partial) result can be provided from the system.</td>
</tr>
<tr>
<td><strong>Load balancing</strong></td>
<td>When operating in a distributed or parallel system, it is convenient that all nodes have (approximately) the same amount of <strong>load</strong>. The term <strong>load balancing</strong> here means both <strong>data</strong> and <strong>routing load balancing</strong>: the former property tells whether all nodes (approximately) manage the same amount of information; the latter depicts that the system is able to route throughout the set of nodes without hotspots (i.e., nodes with high routing process).</td>
</tr>
<tr>
<td><strong>Correctness</strong></td>
<td>When all elements ( e ) from the result ( R ) of a given query ( Q ) are all in the set ( A ) of possible answers for ( Q ) (i.e., ( R \subseteq A )), is said that the result ( R ) is correct and, by extension, the distributed algorithm used to resolve the query. In other words, ( \forall e \in R</td>
</tr>
<tr>
<td><strong>Completeness</strong></td>
<td>This property is somewhat complementary to the property above, in such a way that <strong>correctness</strong> does not ensure that all elements matching the query are returned in the result set. Thus, a distributed algorithm is said to provide <strong>completeness</strong> in its results if and only if ( R \supseteq A ). Note that ( R ) can contain some elements that actually are not part of the expected answer ( A ).</td>
</tr>
<tr>
<td><strong>Intersection</strong></td>
<td>This property of any kind of query, results provided to the querying node can appear already selected with only those items that really are interesting for the user. In this case, the <strong>intersection</strong> of the found data with the query’s predicate has been done on the system side. Otherwise, results needs a last step in the user side to prune all items that actually are not useful to the user.</td>
</tr>
</tbody>
</table>
Other systems like CAN [5] employ instead a multi-dimensional keyspace, where nodes are identified by a vector of values of the form \{id_1, id_2, ..., id_M\}. But, do application data objects coincide with the supported SPS keyspace? Unfortunately not. Application data domains (e.g., geographical location of services) do not resemble SPS keyspaces (e.g., Chord keyspace) with respect to dimensionality, type of information and/or data domain.

Therefore, SPS require mapping techniques to support data object indexing, that convert and adjust application data domains to the SPS keyspace. In addition, this mapping must be deterministic and common for all nodes, so that for the same data object any node will produce the same key. Examples of mapping techniques are hash functions like SHA-1, space filling curves (SFCs) like Z-curve [7] or Hilbert Curves [8], or based on the iDistance [9]. Traditionally SPS make use of uniform hash functions to distribute information uniformly at random between nodes for load balancing purposes. Their drawback is the loss of data locality. That is, two near data objects in the application domain will be randomly placed into the system, so that close nodes will not store similar data objects. Similarity queries cannot be efficiently deployed in this scenario. The naive solution, thus, would be broadcasting the similarity query to all nodes. This approach does not scale for large distributed systems and motivates the application of other kind of techniques thereafter.

Order-preserving and locality-preserving hash functions, as well as SFCs, deal with this locality problem in some degree and are the most deployed in current solutions. By its utilization, indexed data by nodes retain a certain data locality, so that the system efficiency notoriously improves. Therefore, varying either mapping technique or application data domain, the performance of the system will differ from solution to solution. This motivates the inclusion into the evaluation framework of both dimensionality and mapping approaches. By dimensionality we refer to the ability of the given solution to deal with either multi- or one-dimensional application data domains, and by mapping
approach the way that the dimensionality reduction is performed. We do not consider the
details of the mapping techniques regarding to type of information and domain because
ythey are application-dependant and fall out of our study.

3.2. Quality of Service Criteria

Quality of service issues are differentiated into two sets. The first one considers evalua-
tion parameters for the insertion and search operations. The last one considers the quality
of the result set of the query resolution.

3.2.1. Insertion and Search Evaluation Criteria

In order to unify the terminology along the chapter, we do not make difference between
data item or index, and so they are referred to as data object. Let us suppose now that a
user is saving the information of its business and location, namely data object, into the
distributed geographical service location application. The corresponding node inserts the
data object into the system and, this way, it becomes available to all nodes. By doing
this, we now consider three properties related to the processes from data insertions to
query resolution. Firstly, should we consider the amount of storage used for each data
object insertion? The amount of data object copies is considered because it measures the
system’s stress for data storage. Intuitively, storing a single copy of a data object would
be sufficient to find it in future searches, and actually this is the best case. For instance,
traditionally DHTs introduce data objects only once. But either the specific application
data domain or the SPS design can pose an obstacle, requiring to adopt a more complex
approach. We summarize this into the storage efficiency property. Note that we do no
consider replication and caching schemes in this study. Actually, they are applied to a
wide variety of distributed and parallel systems, what includes the systems analysed in
this chapter.

Another property included in the evaluation criteria is the time efficiency. The re-
sponse time for data objects insertions and search resolutions becomes the most visible
and detectable to the end user. Long-lasting operations cause desperation to human users
and must be avoided. In particular, because we are evaluating similarity queries, we only
account the time required for query resolution under time efficiency. Even when a solu-
tion reaches an adequate degree of storage and time efficiency, the solution can be unfair
among nodes. For instance, some nodes could receive a greater amount of data objects
than the rest, and some nodes could also be used for routing purposes more frequently
than others. A solution will be more scalable when it provides both data and routing load
balancing.

3.2.2. Evaluation Criteria of the Result Set

Now, let us suppose that a user has performed a range query \( Q \) in our distributed ge-
ographical service location application example from a certain node, and this querying
node is already in possession of the result set \( R \). Let us suppose that we also know the
exact set of data objects \( A \) expected to recover from \( Q \). How can we measure the quality
of \( R \)? To this end we introduce two other properties: query correctness and completeness.
Broadly speaking, a query \( Q \) is resolved with correctness, when the result \( R \) only
contains data objects such that they were expected (i.e., all data objects from \( R \) also ap-
Note that a correct \( R \) may not contain all the expected data objects (formally \( R \subseteq A \)).

As long as a query resolution provides the results with completeness, this ensures that \( R \) contains at least all elements of \( A \). This allows however that some useless data object could be retrieved during the query resolution (formally \( R \supseteq A \)). In such cases, querying node has to prune the results to leave only useful elements before passing them to the user. This is what we call a user-side prune. Otherwise, a system-side prune has detected all useless data objects and removed from result set if any. We consider the way that information is pruned into the parameter intersection.

Moreover, it is easy to demonstrate that providing both correctness and completeness is a sufficient and necessary condition for a query to provide all possible results. That is, from correctness we have \( R \subseteq A \), and from completeness \( R \supseteq A \) (see Table 2). So, there is only a possibility that makes both conditions come true: \( R = A \). But when a system resolves queries providing both correctness and completeness, the operations could become too expensive either on time or storage.

### 3.3. Parallel Computing Evaluation

Broadly speaking, the consequence of query parallelization in a distributed setting is twofold: the query is resolved in a shorter response time, and the usage of system resources increases. In particular, given that SPS perform node inter-communication by message passing, the bandwidth usage increases between all participant nodes, as well as the usage of nodes’ resources (e.g., computing cycles and main memory). Given that SPS provide an inherent way of parallelizing tasks, we provide in this section an analysis on the parameters from the evaluation framework, which determine the feasibility and efficiency of the parallel computing. Parameters appear detailed in order of significance.

The most important factor is the system’s topology. The way that nodes are organized dictates if operations can be performed in parallel and how. For instance, we have depicted two algorithms for range queries, depending on whether the topology provides path convergence or not (see Algorithms 1, 2). Remember that topologies with path converge provides no utility on query routing parallelization, since all sub-queries will travel along the same path.

Given a certain topology, systems can implement several algorithms to resolve a kind of query. That is, different search algorithms provide a solution to region- or sphere-based range queries, as well as to k-NN queries and spatial queries. It is easy to see that algorithm’s efficiency and adequacy to the system greatly decide the performance of the query parallelization.

Some properties of the system limit also the performance on the query resolution. As it is stated before, path convergence prevents parallelization because it wastes useful resources. The amount of data storage, like the number of object copies, can provide several ways of accessing to the data, for instance. This facilitates that queries are parallelized. In addition, time delay in node inter-communication is not negligible. Implementing a parallel query resolution reduces the response time and increments the system’s performance.

The last property we consider is the system’s load balancing. The reader could see this property as uninteresting for the object of analysis. Actually, data and routing load balancing is very influencing. Whether data is uniformly stored into nodes or its
Table 3. Tuning of the evaluation criteria

<table>
<thead>
<tr>
<th>Evaluation criteria</th>
<th>Tuning 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topology</td>
<td>Descriptive name</td>
</tr>
<tr>
<td>Dimensionality</td>
<td>1-D</td>
</tr>
<tr>
<td>Mapping approach</td>
<td>1:1</td>
</tr>
<tr>
<td>Storage efficiency</td>
<td>Num. data copies</td>
</tr>
<tr>
<td>Time efficiency</td>
<td>Num. overlay hops</td>
</tr>
<tr>
<td>Load balancing</td>
<td>Yes</td>
</tr>
<tr>
<td>Correctness</td>
<td>Yes</td>
</tr>
<tr>
<td>Completeness</td>
<td>Yes</td>
</tr>
<tr>
<td>Intersection</td>
<td>System side</td>
</tr>
</tbody>
</table>

1 The term N/A can appear everywhere when the tuning is not applicable for a given system.

distribution is skewed among nodes determines the query’s response time, making it stable or highly variable, respectively. Alternatively, routing load balancing will provide less overhead on query routing and, thus, less overhead on query processing to participant nodes.

Summing up, the reader should expect to read about systems that greatly differ from each other. Not only the topology, but also a long list of system parameters influence the corresponding costs as well as the performance of parallel computing.

3.4. Evaluation Criteria: Tuning and Terminology

All parameters introduced before are considered within the evaluation framework for all systems. Actually, systems considered into this chapter were designed so that, given a kind of query and an application context, they provide a trade-off between all these parameters. To put an example, a system that supports k-NN queries, could incur in more storage usage in only some peers (load unbalancing), in order to provide low response time (time efficiency) in system’s operations, while the result set remains both correct and complete.

Table 3 shows the common measurements used in this chapter to value the considered systems. Note that in both storage and time efficiency measurements, 1 is the best case. The best case for data storage is when only a data copy is introduced into the system (no replication or caching techniques are considered), and for time efficiency, when the response is at only one overlay hop. For the sake of simplicity, the denominator is only shown on the comparison tables.

In addition, before starting with the system evaluations, let us detail the common terminology we will use along this chapter. Note that every work uses its own nomenclature, but here for clarity to the reader, we unify them as much as possible to a common and simpler naming. The reader can find it in Table 4. Note that selectivity ratio expression refers to the ratio of the data domain that the query is selecting from. For instance, let us suppose an arbitrary range query with a selectivity ratio of 50% on the application data domain or on the i-th attribute. This means that the query is demanding for objects living on a half of the application data domain or the attribute domain, respectively. In the following sections, we delve into the comparison of different distributed systems.
Table 4. Common terminology along system evaluations.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Number of nodes within the network</td>
</tr>
<tr>
<td>$M$</td>
<td>Number of attributes of the application data domain</td>
</tr>
<tr>
<td>$M_D$</td>
<td>Number of attributes included in the data object</td>
</tr>
<tr>
<td>$M_Q$</td>
<td>Number of attributes included in the query</td>
</tr>
<tr>
<td>$S$</td>
<td>Total query’s selectivity ratio</td>
</tr>
<tr>
<td>$S_i$</td>
<td>Selectivity ratio of the $i$-th attribute</td>
</tr>
<tr>
<td>$K$</td>
<td>Number of nodes to visit to complete the query</td>
</tr>
<tr>
<td>$K_i$</td>
<td>Number of nodes to visit for the $i$-th attribute during the query resolution</td>
</tr>
<tr>
<td>$k$</td>
<td>Number of data objects to retrieve in a k-NN query</td>
</tr>
<tr>
<td>$f_o$</td>
<td>Stands for fan-out (i.e., number of outgoing links in the routing table)</td>
</tr>
</tbody>
</table>

4. Supporting Range Queries

After the first SPS appeared on 2001, researchers studied ways of enhancing the SPS by supporting high-level queries. The first kind of search abstraction we study is the range query. We present in Table 5 the measurements from different remarkable systems, using the notations from Table 4. Systems appear in ascending order from the publication year. This provides an outlook on the system design evolution. Hereafter we proceed with the study by classifying their topology into two big sets, namely flat and hierarchical SPS. Differentiating systems from their topology will help to a better understanding of their lookup algorithms and performances.

Actually, it is well-known from software engineering field that an algorithm complexity is tightly influenced by the data structure supporting it. SPS are not an exception. A SPS can be seen as a distributed data structure, so that every node is responsible of only a part of it. Thus, the way nodes are distributed within the network and the connectivity between them will determine the cost of the query resolution.

4.1. Flat Systems

The works considered here employ either a ring-based or a grid-based topology. Thus, algorithms cannot be applied exactly to each other. This motivates their further classification into this study.

4.1.1. Ring-based Topology Systems

The following systems, MAAN [10], SkipNet [4], Mercury [11] and M-Chord [12], distribute nodes in a ring topology. Systems aim at providing a time- and resource-efficient query resolution. This motivates that data objects should be strategically placed within the network. Ring-based topologies (see Fig. 3(a) for an example) provide an easy way of iterating among successors and/or predecessors. This becomes a powerful and efficient mechanism of communication whenever the system stores data objects contiguously within the SPS keyspace. Different mapping approaches appeared to this end. Any given mapping technique has the challenge of placing contiguous data objects in the data
domain, contiguously within the ring. To do so, most of mapping functions are order-preserving hash functions, so that if $f()$ is the function and $x, y$ two data objects, whenever $x \leq y$, $f(x) \leq f(y)$ succeeds.

A sketch of the algorithm used by all these systems is detailed in Algorithm 2 and depicted in Fig. 3(a). A querying node starts the search. Let us assume that the query $q = [lb, ub], lb \leq ub$, is asking for all objects between the lower bound $lb$ and the upper bound $ub$. During the first routing phase, the query is directed to the corresponding node that owns one of these bounds, namely $f(lb)$, and thereafter, the query is spread among all nodes responsible of the range $[f(lb), f(ub)]$. For the sake of clarity, the figure depicts a sequential node visiting order, but more complex algorithms (e.g., [13]) can be applied to achieve that in a more time-efficient way. See Section 4.3 for a discussion on how it can be implemented. Now, let us see what systems do.

The time efficiency column in Table 5 depicts clearly the main two phases on the range query algorithms. SPS studied here locate a node in a logarithmic number of hops with regards to the number of nodes in the worst case and, so, the cost formulation includes a first operand ($\log N$). The second operand corresponds to the cost of spreading the query within the queried range. Generically, this part is considered to be the number of nodes (and so single hops) that are necessary to visit to answer the range query. To put an example, SkipNet has a response time of $O(\log N + S \times N)$. The logarithmic cost is for the first part and $S \times N$ corresponds to the time needed to visit the set of nodes responsible of the queried range. $S$ means the selectivity ratio of the range query, and we assume that all $N$ nodes are uniformly distributed at random.

There are two particular systems with a ring topology, MAAN and Mercury. They show a more complex formulation of the time efficiency. This is because both of them map every single dimension from a M-dimensional data domain into the 1-dimensional SPS keyspace. This motivates the increase on the data storage requirements. Above all, they are in essence the same kind of approach. The difference lies in which MAAN consists of one ring, responsible of all dimensions from the data domain. Instead, Mercury employs as many rings as dimensions is considering the application data domain. Rings in Mercury become clusters of nodes sharing the same information of a given dimension.

These systems employ an order-preserving hash function to determine the responsible node for a given data object. Instead, M-Chord uses iDistance [9] and this mapping approach behaves in a different way. Adopting iDistance requires of few steps. 1) Data objects are sampled before the network creation in order to find out $n$ data clusters. And 2), for every non-overlapping data cluster, a centroid data object and the radius to the farthest object from the same cluster are set up, and become global knowledge for all nodes. Therefore, using iDistance is inappropriate in applications where there is a little or no knowledge of the dataset, and because the system needs of global knowledge, that should be avoided in large scale distributed systems.

In addition, by using the iDistance mapping, range queries are proceeded as in the sphere-based approach. Given that data objects are clustered, a given range query may overlap partially or totally areas from different data clusters. iDistance transforms any overlapping area into a contiguous segment of the SPS keyspace, but all partial segments may not be contiguous among them. This motivates that the range query resolution is parallelized in systems based on iDistance mapping, and in particular in M-Chord. Querying node, then, identifies the queried keyspace segments and sends as many messages as number of segments. The two phases approach still remains here, but for any individ-
4.1.2. Grid-based Topology Systems

SWAM-V [15] and HyVonNe [16] are based on a grid topology. In particular, they place nodes into a Voronoi diagram, and then the dual Delaunay triangulation organizes the neighbours of nodes. HyVonNe selects a leader for every grid’s region in order to make it more flexible to node movements and topology changes. Alternatively, SWAM-V adds some other links to far away nodes to reduce the time latency down to a logarithmic cost. Given that this routing cost is a lower bound for both systems, we only include the full study of SWAM-V.

The key difference between SWAM-V and the other flat ones is the following: nodes in SWAM-V share the information they have, instead of inserting it into other nodes. This fact motivates the organization of nodes according to the content similarity among nodes, and a particular design of the lookup algorithms. The two phases lookup approach remains still valid. The difference resides in that (hyper)sphere-based range queries are considered, instead of (hyper)rectangle or hypercube. This way, the first step locates a node responsible for a central point \( p \); in the second step the query is spread to the nearest nodes (delimited by a time-to-live), and this determines the queried radius. Reader may have observed that this approach does not provide completeness to the result, given that some farther data objects can exist related to the range query. And actually this is what happens. But, by definition of the sphere-based range query (see Table 1), the result set is complete whenever the whole given radius is visited.

4.2. Hierarchical Systems

A hierarchical system organizes nodes in the shape of a pyramid, with each row or level of nodes linked to other nodes beneath them. For the purpose of our analysis, we relax
its definition by allowing nodes appear either at any level on the hierarchy or only in its leaves. In fact, the hierarchy still remains even though only in a logical level. Compared to flat topologies, most of hierarchical ones provide a natural way of performing lookup operations in a parallel way, instead of a sequential resolution. That is, the hierarchical structure provides different paths where to spread the search operation from the very beginning, namely from the querying node (see Algorithm 1 and Fig. 3(b) for an example). As it is stated in Section 3.3, there exists a trade-off between network overhead and time efficiency: This approach incurs in more overhead, for the greater bandwidth usage to say the least, so that the whole result set is retrieved from the system within a shorter response time. Unless noted, all shown time measurements for hierarchical systems consider a parallel lookup resolution. The included hierarchical SPS in this part of the study are broadly classified into two topological groups: tree-based and super-peer-based approaches.

4.2.1. Tree-based Topology Systems

The tree-based topologies are the following ones: Trie-based [14], DST [17], Skip Tree Graphs [18] and SDI [19]. These systems (except DST) place nodes into the leaves of the tree. Then, the logical built tree specifies the interconnection between nodes and, thus, determines the time delay for lookup operations. See Fig. 3(b) for a generic example. Trie-based constructs a load-balanced trie, where all nodes are responsible for the same amount of information. The drawback here is that nodes require from prior knowledge of the dataset distribution to correctly build the trie. One of the strengths is that Trie-based is designed from the very beginning for a real environment, so nodes are placed strategically within the trie to provide structurally data replication. Trie-based provides two algorithms for range queries. The time efficiency shown is for the min-max traversal algorithm, that consists exactly in the sequential, two phases approach explained earlier. An improved approach, called shower algorithm, is also presented and consists in a parallel query resolution, very similar to that shown in Fig. 3(b). This case, the resolution time decreases down to a logarithmic cost, at the cost of a greater network overhead.

Skip Tree Graphs become an abstraction for Skip Graphs/Nets, that provide a tree-like navigation of the distributed data structure. Because Skip Tree Graphs are isomorphic to Skip Graphs/Nets, Skip Tree Graphs are considered to provide the same costs like in SkipNet. Though the hidden constant in the time efficiency is smaller than in SkipNet, Skip Tree Graphs suffers from greater delays at insertion times.

SDI is a particular tree-approach where the navigation, differently from the systems before, is performed in a bottom-up approach. All lookup operations in SDI are inherently parallel, so that the time delay is reduced to a logarithmic cost. SDI, unlike Trie-based, Skip Tree Graphs and DST, does not utilize any mapping function to data objects, but uses the application data domain to organize information into nodes.

There exist a family of systems based on data-driven tree abstraction built upon DHTs, where PHT [20] (Prefix Hash Tree) embodies this family. PHT builds a binary tree link structure among nodes, so that nodes storing contiguous data appear connected. Moreover, this structure is said data-driven because data insertions determine the tree organization. PHT provides data load balancing because it is based on the default DHT functionality \( \text{put}(\text{key}, \text{value})/\text{value} \leftarrow \text{get}(\text{key}) \). But PHT’s penalty is the great overhead in communication cost. PHT resolves range queries iterating over a logical contiguous data domain, but actually along far away nodes from each other. This augments the
total overhead on the routing cost compared to other systems with data locality, providing a poor solution in terms of routing efficiency.

Actually, we include the analysis of DST (Distributed Segment Trees) as a lower bound of PHT and PHT-based systems. DST provides lower time delays than PHT and this is because of the structural data replication performed in DST. Every object insertion is performed in average into $O(\log N)$ nodes. This fact, in addition to the inherent parallel lookup resolution, provides a logarithmic resolution response time in terms of hops. The drawback is that this design clearly does not provide load balancing. Firstly, because higher nodes in the tree receive more queries because of the top-down tree routing. In addition, DST insertion algorithms tries to balance the data load among nodes by defining a maximum storage threshold. Unfortunately, it is easy to see that higher nodes in the tree will reach sooner the threshold than leave nodes, producing a data load unbalancing among nodes.

4.2.2. Super-peer-based Topology Systems

Three systems based on a super-peer topology [21] are considered: the work of Liu et al. [22] (namely LET), EZSearch [23] and JXTA Search [24]. The main key on the design of a super-peer topology is that there are two types of nodes: normal peers and super-peers. Broadly speaking, normal peers store information and are connected to a super-peer. The role of super-peers is indexing information of connected normal peers, in order to speed up lookup resolutions. Super-peers are inter-connected in a way so that super-peers can reach each other with low delay. Evaluating the fairness of the system, it is easy to see that super-peers receive more communication than normal peers and stored information. Thus, super-peer-based designs does not present good qualities in terms of load balancing (as it can be seen in Table 5).

These systems face the problem from two different viewpoints: whilst in EZSearch data is inserted into the system, super-peers on LET and JXTA Search (based on JXTA [25]) build the indexes of the information that their normal peers possess. In addition, while LET and JXTA Search present a two-tier hierarchy (normal peers below and super-peers above), EZSearch (based on ZigZag [26]) builds a balanced tree of variable height, so that each leaf cluster has approximately the same amount of peers. Super-peers in EZSearch (called head and associated-head), can appear in more than one level in the hierarchy. This topological differences clearly determines the time efficiency on range query resolutions.

In these systems, any node can start the range query, and it is directed to a corresponding super-peer. In LET and JXTA Search, the node’s super-peer is the corresponding super-peer and it is responsible of checking into its indexes whether the query can be resolved locally in the cluster, or the super-peer needs to contact to other super-peers. This global knowledge is updated between super-peers by exchanging periodically summaries of their indexes. This way, the query forwarding is more efficient and unnecessary super-peers are not visited. The operation concludes when asked super-peers send to the corresponding super-peer their results and then, all them are forwarded to the querying peer. Note that in JXTA Search nomenclature, super-peer nodes are called hubs, normal peers are differentiated between information providers and consumers, and clusters are called groups.

In EZSearch, range queries are directed to the corresponding super-peer of the corresponding cluster. Unlike LET and JXTA Search, EZSearch makes the assumption that
there is a necessary population of nodes in leave clusters, so that all data objects for the range query will live there. This assumption is guaranteed by correctly setting up the \( z \)-factor, that determines the number of nodes within clusters and, thus, the height of the tree. The goal is clear: avoid visiting other close clusters, that would introduce longer response times. Note that the time efficiency formulation also accounts the \( z \)-factor, and that cost can be interpreted as the number of hops required to contact the corresponding super-peer.

4.3. Range Query Evaluation: Conclusions

Most of systems interpret range queries as a region-based search instead of the sphere-based approach. This is because with the first one, systems can guarantee that whenever there are some data objects lying within the queried range, they will be retrieved. But, actually, these two interpretations of range queries address different problems. For the sake of clarity, while a region-based approach is necessary for geographical location service, in order to provide all elements from the given region, the same interpretation cannot be used in the case of Information Retrieval (IR) applications, or in scenarios where nodes contribute with their own content without inserting it into the system.

Actually, the sphere-based range query can be seen as a k-NN query with a fixed system-wide radius. This is very useful in certain application domains where data object indexation is not so strict, but flexible. For instance, documents in IR applications are indexed according to variable documents meta-data, and nodes supporting their own content can provide a variable number of any kind of data objects. For these scenarios, a region-based approach is not practical in terms of number of peers to be visited and, thus, response time: almost all nodes would be candidate for visiting during the query resolution. This means that every query should be broadcasted to all nodes, which is not suitable in the large scale. Thus, a sphere-based range query deals with the uncertainty problem and provides a feasible solution.

The quality of sphere-based approach depends on two factors: 1) the quality of information and/or node clustering achieved by the system, and 2) the query radius that determines the area to search into. Both of them depend on the specific system design (see SWAM-V [15] or the work of Liu et al. [22] for some examples). Unfortunately, there is a factor that systems cannot govern: the dataset distribution. This factor informs about the data clustering degree. For this kind of systems, highly clustered data objects will provide inherently better results than a dataset that is uniformly distributed within the data domain.

Most of systems, instead, insert data objects into the system and they are stored by the responsible nodes, selected deterministically. This way, an exact match is resolved in the same way than in the insertion case. For these systems the dataset distribution becomes also a key issue. A uniformly distributed dataset will provide inherently good data load balancing among nodes. Otherwise, in most of the SPS, the system will suffer from data load unbalancing, given that some nodes will support more data objects than others. Actually, this problem is an open issue for most of the SPS and it is discussed in Section 7.

We have also explained that hierarchical systems provide in most of them a natural, inherent way of parallel lookups. What about flat topologies? There exists an important reference in this field. The work of S. El-Ansary et al. [13] introduces a broadcast algo-
rithm for ring-based topologies with a total time delay of $O(\log N)$. This algorithm takes advantage from the nodes’ knowledge. Every node in a ring topology knows the fragment of the keyspace it is responsible for, as well as the keyspace fragments of its neighbours. This way, all nodes are contacted by the broadcast message in the worst case with a time delay of $O(\log N)$. It is easy to see that a variation of this algorithm could also be applied for the case of range queries. During the second phase of the range query, a set of contiguous nodes are responsible of the queried range and must be visited. Thus, the first visited node could start a range-restricted broadcast, or multicast, based on the same technique. For instance, SkipNet has a time delay of $O(\log N + S \times N)$. The SkipNet improvement would be from $S \times N$ to $\log(S \times N)$, reducing the response time down to $O(\log N + \log(S \times N))$. However, the trade-off still exists here. Less time delay means greater network overhead.

To conclude this section, it is worth noting that in last years more and more systems aim at providing enhanced lookup services by means of hierarchical SPS, as it can be seen from last rows in Table 5. This is motivated from the fact that most of the hierarchical designs provide inherently parallel query resolution. As we have discussed early in Section 4.2, a parallelized algorithm is not for free, but has the cost of more bandwidth usage and busy nodes per unit time. Nevertheless, given that nowadays Internet connection bandwidth and computer resources are growing rapidly, bandwidth and computation power are considered as not so restrictive as some years ago. As it is seen from our analysis, this opens a door to systems that are not so efficient in resource consumption but effective in reducing response times.

**Notes for Table 5**

- The cost depends on the number of nodes into the system $N$, the number of clusters $|C|$ and the query selectivity ratio $S$.
- Costs inferred from the search algorithm/system description.
- Sequential, two phases resolution cost. The cost of the corresponding parallel algorithm is $O(\log N)$.
- The presented costs are from the parallel resolution algorithms.
- EZSearch makes the assumption that a query is resolved by visiting a single cluster. Here, $C_N$ refers the number of nodes into that cluster.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>JXTA Search [24]</td>
<td>Tree (Super-peers [21])</td>
<td>1-D</td>
<td>NM</td>
<td>1</td>
<td>$\Psi(N,</td>
<td>C</td>
<td>, S)^n$</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>MAAN [10]</td>
<td>Ring (Chord [3])</td>
<td>1-D</td>
<td>1:1</td>
<td>$O(M)$</td>
<td>$O(\log N + K)$</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>User side</td>
</tr>
<tr>
<td>SkipNet [4]</td>
<td>Ring (Skip Graph [27])</td>
<td>1-D</td>
<td>1:1</td>
<td>1</td>
<td>$O(\log N + S \times N)^b$</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>Mercury [11]</td>
<td>Multi-Ring (Symphony [28])</td>
<td>M-D</td>
<td>1:1</td>
<td></td>
<td>$O(MD)$</td>
<td>$O(\sum_{i=1}^{M} (\log N + S_i \times N))$</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SWAM-V [15]</td>
<td>Small-World (Voronoi diagram)</td>
<td>M-D</td>
<td>NM</td>
<td>1</td>
<td>$O(\log N + S \times N)$</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>Trie-based [14]</td>
<td>Binary Tree (P-Grid [29])</td>
<td>1-D</td>
<td>1:1</td>
<td>1</td>
<td>$O(\log N + S \times N)^c$</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>Liu et al. [22]</td>
<td>Tree (Super-peers [21])</td>
<td>M-D</td>
<td>NM</td>
<td>1</td>
<td>$\Psi(N,</td>
<td>C</td>
<td>, S)^n$</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>M-Chord [12]</td>
<td>Ring (Chord [3])</td>
<td>M-D</td>
<td>M:1</td>
<td>1</td>
<td>$O(\log N + S \times N)^b$</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>DST [17]</td>
<td>Binary Tree</td>
<td>1-D</td>
<td>1:1</td>
<td>$O(\log N)$</td>
<td>$O(\log N)^d$</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>Skip Tree Graphs [18]</td>
<td>Tree + Skip Graph [27]</td>
<td>M-D</td>
<td>M:1</td>
<td>1</td>
<td>$O(\log N + S \times N)$</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>EZSearch [23]</td>
<td>Tree (ZigZag [26])</td>
<td>M-D</td>
<td>NM</td>
<td>1</td>
<td>$O(\log N + C_N)^e$</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>SDI [19]</td>
<td>Tree</td>
<td>M-D</td>
<td>NM</td>
<td>1</td>
<td>$O(\log N)^d$</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
</tbody>
</table>
5. Supporting k-NN Queries

$k$-NN queries provide up to $k$ data objects that are the most alike to a given object. The reader can find in Table 1 its formal definition and in Fig. 1(b) an example. Besides, the basic algorithm is detailed in Algorithm 3. A querying node performs the k-NN query by specifying a pair $q = \{p, r\}$ and $k$: $p$ is a point or data object and $r$ defines an initial radius, and $k$ sets the maximum number of elements of the result set.

It is easy to see that sphere-based range queries and k-NN queries are very similar. In fact, the difference lies with the value set to the radius $r$. While in sphere-based range queries it takes a system-wide, fixed value, the value of $k$ is readjusted in k-NN queries (lines 8, 17 of the Algorithm 3). In consequence, any solution that adopts a sphere-based range query approach, can also resolve k-NN queries easily after minimal modifications to the range query algorithm. In particular, most of the systems included in this section appeared also in the range query evaluation section. Some other systems are appended.

It is worth noting that most of time efficiency measurements of Table 6 have the shape $O(\log N + K)$. The first part $O(\log N)$ considers the time to reach the corresponding node. From Table 4, $K$ means the number of nodes required to be visited to complete the query, but its value is hard to quantify formally. Thus, $K$ express a supplement in the response time on the query resolution. Some factors influence $K$‘s exact value: 1) the system design, 2) the number $k$ of elements to retrieve, and 3) the specific situation of the network while the query is performed. The last factor includes either the node organization, as well as the amount of data stored in nodes. It is easy to see that while less data objects are stored in nodes, more nodes will likely be visited. Similarly, when the value of $k$ increases, the probability to visit a higher number of nodes also augments.

Hereafter we proceed with the study of a remarkable set of systems, differentiating between flat and hierarchical SPS. The topology greatly influences into the algorithm design and its performance.

5.1. Flat Systems

We broadly differentiate the systems between ring-based and grid-based topology systems. Algorithms and techniques are different in either case.

5.1.1. Ring-based Topology Systems

We consider in this section the following systems: M-Chord [12] and AON [30]. M-Chord has been discussed in the section before. AON (that stands from Attributed-based Overlay Networks) supports both range and k-NN queries. The motivation to include it into this study is for the given similarity to Mercury. Both systems have essentially the same network structure: one ring for every dimension. The difference lies with the data management. Data objects in AON are neither inserted nor mapped, while nodes in Mercury insert data objects into the network after their mapping. This provides AON for a data-driven ring construction. Nodes in AON are organized into attribute-based rings. The data objects that nodes store locally, specify the node’s place into the ring.

For fast navigation through the ring, AON nodes have a routing table per ring that resembles a Chord finger table. Even though the network structure seems to provide a good infrastructure for efficient query resolution, it poses some inconveniences. Whenever the local information of nodes become dynamic and highly variable, the placement of nodes...
into the different rings they are participating in will vary accordingly. To do so, nodes employ a leave-join mechanism. In consequence, not only the network can suffer from churning (i.e., nodes leaving and joining the network), but also along the time a node is present into the system, since node’s content will certainly vary. Another inconvenience is the amount of node state AON and Mercury require for an efficient routing into all rings that nodes are participating in. AON only presents an evaluation for 1-dimensional scenario, but it is said that multi-dimensional data domains can be also supported.

5.1.2. Grid-based Topology Systems

This part of the k-NN study considers SWAM-V [15], a Voronoi-based node organization. The algorithm applied here is the same like in the case of range queries, but slightly modified to readjust the radius \( r \) in order to complete the query with \( k \) data objects.

Besides, we study two systems based on CAN [5]. They are pSearch [31] and M-CAN [32]. Both systems have a common design decision: reducing the data domain dimensionality (M) to that of the underlying CAN (P). Moreover, they avoid the application of some dimensionality reduction function. This makes easier the use of CAN and accelerates all operations by preventing the use of mapping functions.

pSearch is based on eCAN [33], an improved version of CAN that achieves a worst case logarithmic communication cost between any two nodes. pSearch is intended for Information Retrieval applications, and introduces two algorithms based on eCAN and Information Retrieval techniques. One is called pVSM, for the case of Vector Space Model, where the \( m \) most descriptive terms from documents are taken. The other one is called pLSI, for the case of Latent Semantic Indexing, which reduces the dimensionality of VSM technique to \( l \), \( m > l \), by applying semantic reduction techniques. The reader can observe in the time efficiency formulation a variable cost \( \zeta(k, m) \). This cost summarizes the number of nodes needed to be visited by an arbitrary k-NN query, that highly depends on the number of elements to retrieve \( k \) and the size \( m \) of the vector space model. When Latent Semantic Indexing is used, the delay becomes \( O(\log N + \zeta(k, l)) \).

M-CAN employs the iDistance [9] mapping technique to index all data objects into CAN. M-CAN is then analogous to M-Chord, and the whole M-Chord discussion can be applied to M-CAN. Note that the time efficiency accounts the communication cost based on CAN. It would be easy to replace CAN by eCAN, so that the time efficiency would be reduced to a logarithmic cost \( O(\log N + K) \).

5.2. Hierarchical Systems

Under the umbrella of hierarchical systems we look at two tree-based topology systems, PIRD [34] and SDI [19], as well as to super-peer based topology systems, namely Liu et al. [22] and EZSearch [23]. All systems except PIRD have been discussed in the range query section. Given that these systems employ a sphere-based range query approach, they modify the query algorithm to successively adjust the radius \( r \) in order to support k-NN queries. Let us put the focus on PIRD.

PIRD is based on Cycloid [35] and structurally provides geographical k-NN queries, where the result set contains indexes to data objects that are located in nodes close to the querying node. This way, querying peer will accelerate the last peer-to-peer transmission. To do so, Cycloid introduces a two-layer indexation. The first layer organize nodes and indexes into different clusters, and the second layer (i.e., any one of the clusters), organize
information according to the geographical proximity. The proximity is presented as a distance vector obtained from a sampling to a set of landmarks. This way, the distance evaluation is performed by comparing those vectors. Nodes also use these vectors to locally cluster the indexes according to the distance. The query algorithm takes advantage of this indexing to retrieve the $k$ closest data objects to the query, and also by retrieving them from the geographically nearest nodes to the querying node.

Nevertheless, the indexes are inserted in a system-wide number of times ($L$), incrementing the data storage on nodes. To do so, PIRD builds $L$ different keys in a way that similar data objects will be assigned to similar keys with high probability. This can be seen also as to having similar data objects indexed in the same node. Queries are spread to $L$ different keys, and the querying node has the last responsibility to return a valid result set of $k$ data objects from those $L$ partial sets received from queried nodes. Lastly, the time efficiency shown for PIRD is the time for a single lookup operation in Cycloid, because the k-NN query is parallelized.

5.3. k-NN Query Evaluation: Conclusions

It is easy to see a common denominator within the study of systems supporting k-NN queries. Because of the nature of k-NN query algorithms, systems that also supports range query in a sphere-based approach prevail over the others. Actually, this section does not include any system that supported range queries in a region-based approach. This is because both approaches are quite distant from each other, and pose different challenges during the system design.

Another common characteristic among systems is the two phases approach. Remember that in the first phase the corresponding node is reached, while in the second one close nodes are accessed, in order to build a result set. This resolution mechanism is dictated by the definition of the k-NN query. There are only few exceptions to this though. One example is the innovative design of PIRD [34] that, by means of several keys per data object, the system achieves a per-node index clustering, so that the query resolution is performed locally at several corresponding nodes.

Another family of systems that perform only data indexation are data networks. Nodes in data networks hold their own information and only indexes are built throughout the system. The way the indexation is realized will decide whether the two-phase approach remains still valid, or ad-hoc algorithms will have to face the problem. An example within the family of data networks is the work of Liu et al. [22]. Unless in hierarchical systems supporting range queries, there is not much flexibility for systems to totally parallelize k-NN queries. That is, with a perfect knowledge of nodes’ content, querying node would be able to direct the query to the corresponding nodes. However, this requires of global knowledge, what is not feasible on the large scale. A slight longer response time is the low price users have to pay for.

Notes for Table 6

- $a$ The final delay is related to $\zeta(k, m)$, where $k$ is the k-NN factor, and $m$ is the number of terms in the Vector Space Model.
- $b$ The cost depends on the number of nodes into the system $N$, the number of clusters $|C|$ and the number of objects to retrieve $k$.
- $c$ Queries in PIRD are parallelized, where $O(d)$ is the cost of a single lookup.
- $d$ The presented costs are from the parallel resolution algorithms.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pSearch [31]</td>
<td>Hypercube (eCAN [33])</td>
<td>M-D</td>
<td>M:P</td>
<td>1</td>
<td>$O(\log N + \zeta(k, m))$</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>SWAM-V [15]</td>
<td>Small-World (Voronoi diagram)</td>
<td>M-D</td>
<td>N/M</td>
<td>1</td>
<td>$O(\log N + K)$</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>Liu et al. [22]</td>
<td>Tree (Super-peers [21])</td>
<td>M-D</td>
<td>N/M</td>
<td>1</td>
<td>$\Psi(N,</td>
<td>C</td>
<td>, k)$</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>M-Chord [12]</td>
<td>Ring (Chord [3])</td>
<td>M-D</td>
<td>M:1</td>
<td>1</td>
<td>$O(\log N + K)$</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>M-CAN [32]</td>
<td>Hypercube (CAN [5])</td>
<td>M-D</td>
<td>M:P</td>
<td>1</td>
<td>$O(P \sqrt{N} + K)$</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>AON [30]</td>
<td>Ring</td>
<td>1-D</td>
<td>N/M</td>
<td>1</td>
<td>$O(\log N + K)$</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>PIRD [34]</td>
<td>Hierarchical (Cycloid [35])</td>
<td>M-D</td>
<td>M-P</td>
<td>$O(L)$</td>
<td>$O(d)^c$</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>User side</td>
</tr>
<tr>
<td>EZSearch [23]</td>
<td>Tree (ZigZag [26])</td>
<td>M-D</td>
<td>N/M</td>
<td>1</td>
<td>$O(\log_2 N + C_N)$</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>SDI [19]</td>
<td>Tree</td>
<td>M-D</td>
<td>N/M</td>
<td>1</td>
<td>$O(\log N)^d$</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
</tbody>
</table>
6. Supporting Spatial Queries

Spatial queries can be seen as a particular case of range queries, either in a sphere-based or region-based approach. Supporting spatial queries means that the systems must support operations in at least two dimensions, which specify the geographical location of objects, for instance by a pair \( \{ \text{latitude}, \text{longitude} \} \). In addition, these systems can support either point data objects or rectangular data objects. The former describes the data object’s location by a single point \( \{ x, y \} \), while rectangular ones are extended objects that occupies an area. Rectangular data objects are usually specified by a pair of points \( \{ x_1, y_1 \}, \{ x_2, y_2 \} \) which describe the minimum bounding rectangle of the covered area.

To give some examples, a restaurant’s location is seen as a point data object, while the area where a superstore makes home deliveries can be described by a rectangular area.

When users perform spatial searches, the systems have to provide data objects that are located in or overlap the specified area of arbitrary size, while the search area is usually specified by the minimum bounding rectangle \( \{ x_1, y_1 \}, \{ x_2, y_2 \} \). It is easy to see that the challenges of this kind of systems are particularly different from those ones supporting range queries, what has motivated their study separately.

This section goes on with the evaluation of significant systems supporting spatial queries, by differentiating between flat and hierarchical systems. All systems included here can be found in Table 7, listed in ascending order of the publication year.

6.1. Flat Systems

We consider two systems in this section, SDS [36] and Spatial Query [37]. SDS is a layered design, using Chord as a distributed infrastructure to support a quadtree. Given that the emphasis is put on the quadtree design, SDS is further detailed in Section 6.2.

Spatial Query is designed onto CAN [5]. CAN naturally supports multi-dimensional domains, so that little effort is required to use CAN for supporting spatial queries. Spatial Query supports both point and rectangular data objects. Thus, the focus is moved to guarantee data load balancing through nodes, what is addressed in two ways. A new node selects randomly to join into an area with high data density, and splits it into two sub-areas. Secondly, by limiting the number of times a region can be split. This way, nodes are always responsible of a minimum area. This approach has a double effect though. Firstly, nodes in Spatial Query are distributed more uniformly because of the threshold. But in contrast, nodes responsible of areas very populated that have reached the threshold, have no way of balancing the data load with other nodes.

6.2. Hierarchical Systems

Hierarchical systems are classified in this section into three topological types: tree-based, super-peer-based, and hierarchical topology systems. The last class presents a system that is actually a hierarchy of peer-to-peer nodes that do not meet the topology characteristics from the two first types.

Tree-based Topology Systems

We analyse two tree-based topology systems, DHR-Tree [38] and SDS [36]. Both of them present a distributed design of already existing tree data structure.
DHR-Tree, that stands from Distributed Hilbert R-Tree, is based on P-Tree [39] and distributes nodes through a Hilbert R-Tree. Mainly, a Hilbert R-Tree uses the Hilbert [8] space filling curve to produce the so-called Hilbert values, and use them to index data objects into a B+-Tree. The idea behind this distributed design is to use the Hilbert values also as node identifiers, thus operating as data nodes in a B+-Tree, and to use the B+-Tree as nodes’ routing table. The B+-Tree is populated with data that nodes locally host and the B+-Tree is partially built on every node. Thus, data updates may alter the minimum bounding rectangle a node is responsible for, and so forth on nodes at higher levels in the B+-Tree. The time efficiency in this scenario is tight to the order $d$ of the R-Tree and to the search algorithm of two phases.

SDS (Spatial Data Service) does not build an SPS, but employs Chord as its underlying communication network. In addition, SDS builds multiple indexes for data objects. Thus, the focus is put on balancing the data load among nodes and structurally avoiding communication between neighbouring nodes for updating minimum bounding boxes. To do so, nodes considers the whole data domain managed by a MX-CIF Quadtree [40]. The centroids of every region in the quadtree become, after applying a uniform hash function (like SHA-1), the key to route within Chord. This way, either rectangular data objects or spatial queries are processed locally as in the quadtree. The result of this process is the set of regions in the quadtree where the data object should be indexed or the search be performed, respectively. The next step is routing the corresponding messages to the keys produced by the centroids of the set of regions. Thereafter, nodes responsible of the keys will perform the corresponding operation.

SDS balance the data among nodes by applying two techniques. The first one is the use of a uniform hashing function for the centroids. The second one is by ensuring that data objects are indexed into the quadtree only under a certain level (so-called $f_{min}$) of the tree. Thus, $f_{min}$ dictates the trade-off between load balancing and indexing storage. When spatial queries are performed, the same data index can be retrieved several times from SDS because there is no coordination between nodes to collect a common result set. It is easy to see that this produces a non-negligible network overhead. SDS faces the problem of duplicate data transmissions by limiting at owner nodes that querying nodes could download only once each requested data object. Further requests are discarded. Even though the approach is nice, but quite naive, the time efficiency for spatial queries is reduced to a logarithmic cost. However, the query is parallelized into several keys per query, suffering from more network overhead.

**Super-peer-based Topology Systems**

This section considers two systems: Liu et al. [22] and Globase.KOM [41]. Given that the work of Liu et al. has already discussed in prior sections, we center the study on Globase.KOM and the comparison between them. Both systems employ a similar model of super-peer network, where super-peers maintain an index of all the content that their normal peers hold. Globase.KOM defines a node identifier with three separated parts: the $\{lat, lng\}$ coordinates of the node’s location, the area that the node is responsible for, and a random part to allow several peers in the same geographical area.

A key difference between the work of Liu et al. and Globase.KOM is the number of levels in the hierarchy. While Liu et al. define a 2-layer network structure, Globase.KOM is organized in several layers so as not to overload super-peers. When a super-peer $sp_1$ is overloaded, a new sub-cluster is built under $sp_1$ and a new super-peer $sp_2$ is elected.
The effect is that a set of normal peers that were children of $sp_1$ are now children of $sp_2$, and $sp_2$ remains as a child of $sp_1$, thus balancing the load among super-peers. The network construction algorithm produces a network structure of super-peers and normal peers that resembles to that of an R-Tree. This motivates that the response time for query results is logarithmic in the number of clusters, plus the cost of asking to normal peers. Note that searches are performed in parallel, thus increasing the bandwidth usage.

Hierarchical Topology Systems

This section presents the study of Geophony [42]. Geophony enhances Symphony [28] by means of Cyclone [43] in order to produce a hierarchy of disjoint clusters. Every node appears in all levels it exists, where the top cluster contains all nodes. Thus, it is easy to see that the Geophony’s network structure does not resemble a tree nor a super-peer-based network structure. Nodes have the same responsibilities in all levels they appear. The key point in the Geophony construction is that lower clusters hold nodes of the closer geographical areas.

Data objects are inserted into Geophony according to their geographical location, though it does not supports rectangular data objects directly. To do so, Geophony embeds the geographical location into the node identifier as well as the key of the data object. So that the routing scheme directs the insertion/search operation to the corresponding cluster. While the responsible node is visited for a data insertion, the spatial search is multicasted to nodes of the queried geographical area.

Geophony’s aproach provides a double proximity. Not only are nodes clustered by their geographical location, but also data objects. In consequence, Geophony provides data locality and fault isolation. Data locality is provided by clustering data from close geographical locations, as well as nodes from the same area. The price of providing data locality is however data load unbalancing. In addition, if a connection error to Internet occurs in an Internet Service Provider, the disconnected geographical areas will still remain consistent: Nodes from the given areas rest connected within the cluster and they can search for content from their area and sub-areas (sub-clusters).

6.3. Spatial Queries: Conclusions

Spatial queries pose additional challenges to systems, like supporting rectangular data objects, even though not all above seen solutions support them. Actually, systems are tuned up to improve their performance in the given geographical application context.

Besides, there is an evolution on the system designs from those supporting range queries to these ones supporting spatial queries. Even though range queries and spatial queries have a lot in common, hierarchical systems are significantly predominant in the study of spatial queries while the same is not true for the study of range queries. This is motivated because a hierarchical approach suits better than a flat one for supporting spatial queries.

Another important issue addressed by Globase.KOM and Geophony is that data objects and nodes are indexed/placed in the geographical area they belong to. This is a very interesting approach that provides at the same time data locality and consistency to the system against system failures. Another consequence is that while clusters organize the information from the same geographical area, most of the queries will have a local effect on the cluster. This provides for the system isolation from traffic from other parts of the network, as well as a likely greater speed up in the network communication.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Spatial Query</td>
<td>Hypercube</td>
<td>2-D</td>
<td>1:1</td>
<td>$O(o_S \times O)^a$</td>
<td>$O(\sqrt{\frac{N}{o_S}} \times O)$</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>Liu et al. [22]</td>
<td>(CAN [5])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DHR-Tree [38]</td>
<td>Tree (Super-peers [21])</td>
<td>M-D</td>
<td>NM</td>
<td>1</td>
<td>$\Psi(N,</td>
<td>C</td>
<td>, S)^c$</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>Globase.KOM [41]</td>
<td>Tree (P-Tree [39])</td>
<td>M-D</td>
<td>NM</td>
<td>1</td>
<td>$O(\log_d N + S \times N)^d$</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>System side</td>
</tr>
<tr>
<td>SDS [36]</td>
<td>Tree (Quadtrees [40])</td>
<td>2-D</td>
<td>M:1</td>
<td>$O(o_S \times f_{min})^a$</td>
<td>$O(\log N + f_{max} - f_{min})^f$</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>Geophony [42]</td>
<td>Hierarchical</td>
<td>3-D</td>
<td>M:1</td>
<td>1</td>
<td>$O(\log</td>
<td>C</td>
<td>+ S \times C_N)^e$</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*a* $o_S$ specifies the ratio of the covered data domain for rectangular data objects.

*b* $O$ refers to the application data domain.

*c* The cost depends on the number of nodes into the system $N$, the number of clusters $|C|$ and the query selectivity ratio $S$.

*d* $d$ is the order of the R-Tree.

*e* $|C|$ is the number of clusters, and $C_N$ refers to the number of nodes of the target cluster.

*f* $f_{min}$ is the minimum level in the quadtree from where insertions are done. $f_{max}$ is the maximum level of the quadtree.
7. Open Issues and The Future on High-level Queries in Peer-to-Peer Systems

Before concluding this chapter, we want to discuss about what we consider the most important open issues in the field of high-level queries in peer-to-peer-enabled distributed and parallel systems. In particular, we introduce the existing trade-off between data locality and load balancing. The transverse problem of the high dimensional data domains is also stated. And finally, we consider the study of peer-to-peer-based data networks as the hot topic for next practical and broadly deployable distributed large scale applications.

Data Locality vs Load Balancing

In dynamic environments, where users perform lots of data operations (insertions, deletions and lookups) as well as where users remain connected only for a certain period of time, fair data distribution in peer-to-peer systems is challenging. Indeed, the problem in the context of this chapter is twofold. Firstly, data objects should be placed with some data locality guarantee, so that similarity queries turn efficient. And secondly, at the same time nodes should afford approximately the same amount of data load. This way, note that not only nodes could be overloaded with too many data objects, but also with lookup processing with high probability.

We can classify the SPS into two big sets. One set formed by systems that require of mapping techniques to support data object indexing, so as to convert and adjust application data domains to the SPS keyspace. The other set is formed by systems that operates with the specific application data domain. Traditionally SPS make use of uniform hash functions to map and distribute information uniformly at random between nodes for load balancing. The mapping drawback is the loss of data locality though. To guarantee a certain data locality, order-preserving or locality-preserving hash functions appeared. For instance, space filling curves (SFCs) like Z-curve [7] or Hilbert Curves [8] embody the set of locality-preserving hash functions.

Unfortunately, even when systems employ mapping functions they are not guaranteeing load balancing, given that the mapping values retain the dataset distribution and, therefore, explicit load balancing techniques must be designed. In addition, the mapping values usually serve only to deterministically specify the location of the object. Mapping values can serve to perform a coarse-grained filtering, but cannot be used for fine-grained filtering like in k-NN queries. This is because most of the descriptive information from data objects has been lost during the mapping process.

We believe that the peer-to-peer systems should support the specific application data domain. This is even more beneficial because the same information is self-indexed, and system does not waste time in making pre-processes (like mapping data objects or queries to SPS keys), accelerating and improving the whole solution. The challenge in this context is moved to balance the data load in order to guarantee fairness in node responsibilities. There are significant works that deal with this problem [44,45]. But, given that most of peer-to-peer-based systems have a diverse topology, systems have to design their particular algorithms to explicitly balance the load.

High-dimensional Data Domains

It has been widely demonstrated that the concept of proximity in high dimensional space may not be very meaningful [46]. These results show that for certain classes of commonly
used similarity functions such as the $L_p$-norm in the Euclidean space, the nearest and furthest neighbor are of the same relative distance to the query point for large classes of data distributions. The lack of relative contrast in terms of similarity is somewhat undesirable, since it is not clear whether the nearest neighbor is meaningful under such circumstances. Thus, the distance function becomes unstable. Some direct consequences of this instability is the inclusion of far away data objects during the query resolution. This overloads nodes in time consumption and usage of computer resource because of filtering far data objects, producing greater response times on data lookups.

Even though this problem is not properly from SPS, these systems experience inefficiency when high-dimensional data domains are supported. This motivates the adoption of advances in the similarity calculation from other data management fields, like database systems. To put an example of SPS permeability, we have seen in this chapter some SPS that were based on tree data structures, like binary trees, B+-trees and quadtrees, for node and information distributed organization. It will not be strange to see new SPS based on last results from the similarity field in high dimensional domains, like algorithms for data clustering [47] or techniques for dimensionality reduction [48].

Peer-to-Peer-based Data Networks

Instead of inserting data objects into the SPS, Peer-to-Peer-based Data Networks (PDN) pose the stress on distributed information indexing while data objects remain locally stored into nodes. This is the way like most of well-known file-sharing peer-to-peer-based applications work actually, like eMule. Thus, the indexed records in essence consist of a pair $\text{index} = \{\text{object}, \text{address}\}$, where $\text{object}$ is a reference to the data object being indexed, probably with additional meta-data to help in search operations, and $\text{address}$ refers to the node’s address storing the $\text{object}$. This indexing structure divides search algorithms into two steps: 1) querying peers perform the lookup operation to the distributed indexing data structure, obtaining a set of $\text{indexes}$ from the system, and 2) querying nodes retrieve the related data objects from nodes whose $\text{addresses}$ appear in the result set.

PDN will remain the most deployed application in the future, because its design facilitates users controlling the content stored locally. The price, though, is information indexing in the user’s computer, which is usually a very little and reasonable cost compared to the volume of the data objects. Indeed, we foresee that PDN will become the hot topic for practical and broadly deployable applications. The research focus will then be placed in efficient distributed indexing techniques. In addition, not only should the indexation technique be time- and storage-efficient, but also it should help in improving the efficiency of data object transmissions. This could be achieved providing indexes of data objects that live in the surroundings of the querying node, as PIRD does. The main shortcoming of PIRD is the strict network structure and number of nodes into the system. Actually, whenever PDN designs resemble to the Internet topology, or whenever the system takes advantage of the network infrastructure [49,50], systems will then benefit from node proximity. Proximity can be then measured in any valuable metric, like latency and/or geographical distance, to say the least.
8. Conclusions

This chapter has presented an evaluation of different systems providing similarity queries for distributed, structured peer-to-peer systems. The evaluation is addressed to systems providing range, k-nearest neighbours and spatial queries. The chapter started with the motivation of the necessity of this kind of queries. A set of generic algorithms were introduced, that provide the main steps of the considered lookup operations, emphasizing on the parallelization of tasks. Then, we provided the definition of the evaluation framework.

After that, a total amount of 20 different systems of the last 7 years are considered within this study. Thus, our study provides a wide overview of this kind of systems since the introduction of first structured peer-to-peer systems in 2001. And in particular, this study focuses on the time efficiency of the query resolution. A total amount of 10 properties for every system are collected from the specific works and discussed along this chapter. When necessary, apart from those 10 properties, additional terms are considered, like query parallelization or amount of node status, in order to provide a big picture of systems, and to guarantee a fair comparison.

Broadly speaking, peer-to-peer-based systems that provide similarity queries provide different search resolution performances according to three factors: the topology, the query resolution approach and the query type. In addition, they are not independent variables, but the topology greatly dictates the rest of the parameters.

Topology. In last years more and more systems aim at providing enhanced lookup services by means of hierarchical SPS, as it is seen along the chapter. This is motivated from the fact that most of the hierarchical designs provide inherently parallel query resolution. Instead, most of the flat architectures provide a two-phase search algorithm, where the system routes the search as a single message until a first responsible node is reached, and thereafter the query is parallelized among all responsible nodes.

Query resolution approach. SPS adopt two main query resolution approaches, which range from DHTs that were redesigned to support similarity queries, to distributed data systems that structurally provide them. In general, we can see that for the first case, systems employ order or locality preserving hash functions to map data objects to the SPS keyspace. For the last case, most of the systems index data objects adopting the original data domain without transformation.

Query type. The type of query also determines the reachable system performance. As we have seen in this chapter, region-based range and spatial queries are more parallelizable than k-NN and sphere-based range queries. Moreover, most of the spatial queries are constructed over hierarchical distributed systems, what nicely fits the spatial query resolution algorithm. Thus, the inherent parallelization of tasks shorten the response time, but this is not for free. It has the cost of more bandwidth usage and busy nodes per unit time. Nevertheless, given that nowadays Internet connection bandwidth and computer resources are growing rapidly, bandwidth and computation power are considered as not so restrictive as some years ago.

To sum up, the field of high-level queries requires of more research in the context of scalable, distributed, peer-to-peer solutions. Some of the open issues are described in this chapter, like the trade-off between data locality and load balancing, the high dimensionality problem and the role that peer-to-peer data networks will play in the future. The research however cannot be addressed from a generic viewpoint for all kinds of systems.
and scenarios. Specific problems have specific challenges targeting particular solutions, as we have seen in the field of spatial queries. Therefore, we foresee more work in this area with solutions that will be more resource efficient, shortening response times, and all this by taking advantage of the peer-to-peer-based parallel computing capabilities.

Acknowledgements

This work has been partially funded through project P2PGRID, TIN2007-68050-C03-03, and FPU National Program, AP-2006-04166, of the Spanish Ministry of Education and Science. Authors appreciate the valuable comments and suggestions from Prof. Alberto Montresor and Gianluca Ciccarelli (Università degli Studi di Trento, Trento, Italy), and from Rubén Mondéjar-Andreu (Universitat Rovira i Virgili, Tarragona, Spain).

References


