Metric-Based Similarity Search in Unstructured Peer-to-Peer Systems

Akrivi Vlachou[1], Christos Doulkeridis[1], and Yannis Kotidis[2]

[1] Dept. of Computer and Information Science, NTNU
7491 Trondheim, Norway
{vlachou,cdoulk}@idi.ntnu.no
[2] Dept. of Informatics, AUEB
10434 Athens, Greece
kotidis@aueb.gr

Abstract. Peer-to-peer systems constitute a promising solution for deploying novel applications, such as distributed image retrieval. Efficient search over widely distributed multimedia content requires techniques for distributed retrieval based on generic metric distance functions. In this paper, we propose a framework for distributed metric-based similarity search, where each participating peer stores its own data autonomously. In order to establish a scalable and efficient search mechanism, we adopt a super-peer architecture, where super-peers are responsible for query routing. We propose the construction of metric routing indices suitable for distributed similarity search in metric spaces. Furthermore, we present a query routing algorithm that exploits pruning techniques to selectively direct queries to super-peers and peers with relevant data. We study the performance of the proposed framework using both synthetic and real data demonstrate its scalability over a wide range of experimental setups.

1 Introduction

Similarity search in metric spaces has received significant attention in centralized settings [7,19], but also recently in decentralized environments [14,17,26]. A prominent application is distributed search for multimedia content, such as images, video or plain text. For example, the SAPIR project[1] aims at creating an infrastructure for similarity search, by adopting a fully distributed architecture based on peer-to-peer (P2P) technology [2]. In such a distributed search engine, the objective is to find all objects that are similar to a given query object, such as a digital image or a text document. Objects are represented in a high-dimensional feature space and a metric distance function defines the similarity of two objects. In order to provide effective search over multimedia content, techniques for retrieval based on generic metric distance functions are required.

Existing approaches for P2P metric-based similarity search mainly rely on a structured P2P overlay, which is used to intentionally store objects to peers

[1] http://www.sapir.eu/

A. Hameurlain et al. (Eds.): TLDKS V, LNCS 7100, pp. 28–48, 2012.

by means of a distributed hash table (DHT) implementation [17,26]. The aim is to achieve high parallelism and share the high processing cost over a set of cooperative computers. In contrast, in this paper, we focus on the scenario of autonomous peers that store multimedia content and collaborate in order to process similarity queries over distributed data. The main differentiating factor is that multimedia content remains stored on the peer that owns it, thus respecting privacy requirements, instead of being stored at arbitrary peers as dictated by the DHT of the underlying structured P2P system.

Our work is motivated by the design of a P2P architecture for image retrieval, as proposed in [33]. In more details, content providers are simple peers that keep multimedia content, which is usually user-generated at the peer. Each peer joins the collaborative search engine, by connecting to one of the information brokers that act as super-peers, using the basic bootstrapping protocol. Super-peers are responsible for establishing the search mechanism. Any peer issues similarity queries for multimedia content (images) and the execution of queries (query routing) is performed at super-peer level, thus directing queries to peers with relevant content.

In this paper, we propose a framework for distributed metric-based similarity search that relies on a super-peer architecture, assuming that cooperative peers store and index their data in an autonomous manner. Each peer must be able to process efficiently similarity queries based on its locally stored data. Thus, each peer indexes its local data by using the M-Tree [9]. The M-Tree consists of a hierarchy of hyper-spheres and is one of the most commonly used centralized indexing techniques for searching in metric spaces. When a peer connects to a super-peer, it publishes the set of hyper-spheres stored at the root of its M-Tree to its super-peer, as a summarization of the stored data. The super-peers store the collected hyper-spheres using an M-Tree index, in order to direct queries only to relevant peers efficiently, thus establishing a *peer selection mechanism*. Capitalizing on their local metric index structures, super-peers exchange summary information to construct metric-based routing indices, which improve the performance of query routing significantly. Then, given a range query, this *super-peer selection mechanism* enables efficient query routing only to that subset of super-peers that are responsible for peers with relevant query results.

The contributions of this paper are the following:

1. We propose a framework for distributed metric-based similarity search, under the assumption that the basic indexing method available on peers and super-peers is the M-Tree index.
2. Our framework relies on the construction of metric-based routing indices for similarity queries in metric spaces, over a super-peer architecture.
3. We propose a routing mechanism that selectively routes similarity queries only to those peers and super-peers that store relevant data.
4. We assess the feasibility of out framework, by means of an experimental evaluation, employing both synthetic and real datasets.

In [32], we shortly described our framework for metric-based similarity search in P2P systems. In this paper, we extend this work substantially by presenting

the proposed metric-based routing indices in detail. We also present our novel algorithms for query routing that capitalize on pruning properties to achieve efficiency. Moreover, we discuss maintenance issues, such as updates and peer churn. Finally, we provide a more extensive experimental evaluation.

The rest of this paper is structured as follows: Section 2 provides an overview of related work. In Section 3, the preliminaries are presented, while Section 4 describes the construction of the routing indices based on the exchanged summary information. Then, in Section 5, we present our query routing mechanism. Section 6 deals with maintenance issues. The experimental evaluation is presented in Section 7 and we conclude the paper in Section 8.

2 Related Work

Similarity search in metric spaces has wide applicability in centralized domains. For an overview of relevant algorithms and techniques, see [7,19].

2.1 P2P Metric-Based Similarity Search

Recently, metric similarity search has also attained increased interest in P2P systems [14,17,26,33]. There are two main approaches for facilitating metric similarity search in distributed environments.

The first approach includes systems like MCAN [17] and M-Chord [26], relying on an underlying structured P2P network, namely CAN [28] and Chord [31] respectively. Both techniques focus on parallelism for query execution, motivated by the fact that in real-life applications, a complex distance function is typically expensive to compute. MCAN uses a pivot-based technique that maps data objects to an N-dimensional vector space, while M-Chord uses the principle of iDistance [20] to map objects into one-dimensional values. Data preprocessing – clustering and mapping – is done in a centralized fashion, and only then data is assigned to peers. Relevant to this work, Batko et al. [4] present a comparative experimental evaluation of distributed similarity search techniques. VPT* and GHT* [3] are two distributed metric index structures that are included in the comparison together with MCAN and M-Chord. Later, bulk loading for structured P2P systems has been proposed [13], focusing on peer splits. Recently, the Metric Index (M-Index) [25] has been proposed as a general approach for metric-based data management.

In the second category, peers store data in an autonomous manner and an architecture that supports efficient similarity search using a super-peer architecture was presented in [33]. In SIMPEER [14], P2P metric-based indexing is supported using the iDistance [20] technique. An extension of SIMPEER for recall-based range queries is presented in [15]. In contrast, this paper provides an alternative technique for similarity search in metric spaces, based on a popular metric index (M-Tree) for data access both on peers and super-peers. More importantly, relying on M-trees as routing indices, we present pruning techniques that enhance the performance of query routing.

2.2 Similarity Search in P2P Systems

Apart from the aforementioned research, there exist several approaches for P2P similarity search that do not focus on metric spaces.

In unstructured P2P systems, a general solution for P2P similarity search for vector data is proposed in [1], named SWAM. Peers autonomously store their data, and efficient search is based on an overlay topology that brings nodes with similar content together. However, SWAM is not designed for metric spaces. Content-based similarity search using a hierarchical P2P network is studied in [16]. A P2P framework for multi-dimensional indexing based on a tree-structured overlay is proposed in [22]. LSH forest [5] stores documents in the overlay network using a locality-sensitive hash function to index high-dimensional data for answering approximate similarity queries. Another approach that focuses of semantic content search over distributed document collections is described in [29], where a hierarchical summary index is built over a super-peer architecture. In [12], Datta et al. study range queries over trie-structured overlays.

Most approaches that address range query processing in P2P systems rely on space partitioning and assignment of specific space regions to certain peers. A P2P framework for multi-dimensional indexing based on a tree structured overlay is proposed in [22]. A load-balancing system for range queries that extends Skip Graphs is presented in [30]. The use of Skip Graphs for range query processing has also been proposed in [18]. Several P2P range index structures have been proposed, such as Mercury [6], P-tree [10], BATON [21]. A variant of structured P2P for range queries that aims at exploiting peer heterogeneity is presented in [27]. In [24], the authors propose NR-tree, a P2P adaptation of the R*-tree, for querying spatial data. Routing indices stored at each peer are used for P2P similarity search in [23]. Their approach relies on a freezing technique, i.e. some queries are paused and can be answered by streaming results of other queries. Recently, in [11], P-Ring is proposed as an indexing structure that enables range query processing.

3 Preliminaries

In this section, we provide the necessary preliminaries for our framework. We start with the system architecture, then we present the data model, the query types and a short overview of the M-Tree index. An overview of the symbols used can be found in Table 1.

3.1 System Overview

We assume an unstructured P2P network that consists of N_p peers. Some peers have special roles, due to their enhanced features, such as availability, stability, storage capability and bandwidth capacity. These peers are called super-peers SP_i $(i = 1..N_{sp})$, and they constitute only a small fraction of the peers in the

Table 1. Overview of symbols

Symbols	Description
d	Data dimensionality
n	Dataset cardinality
N_p	Number of peers
N_{sp}	Number of super-peers
DEG_p	Degree of peer
DEG_{sp}	Degree of super-peer
$dist()$	Distance function
$R(q, r)$	Range query
N	Node of M-Tree
p	Representative object of N
$r(p)$	Covering radius of p
T	Reference to child node of N

network, i.e. $N_{sp} << N_p$. Peers that join the network directly connect to one of the super-peers. Each super-peer maintains links to peers, based on the value of its degree parameter DEG_p, which is the number of peers that it is connected to. In addition, a super-peer is connected to a limited set of at most DEG_{sp} other super-peers ($DEG_{sp} < DEG_p$).

3.2 Metric Spaces

In our system, peers that join the network autonomously store their own data. Each peer maintains its own data objects, while the features extracted from the objects are represented as d-dimensional points. Thus, each peer P_i holds n_i d-dimensional points, denoted as a set S_i ($1 \leq i \leq N_p$). Assuming horizontal data distribution to the N_p peers, the size of the complete set of points is $n = \sum_{i=1}^{N_p} n_i$ and the dataset S is the union of all peers' datasets S_i ($S = \cup S_i$). Notice that our techniques are applicable also in the case of peers storing overlapping data, i.e., $S_i \cap S_j \neq \emptyset$.

Definitions and Query Types. Similarity search in metric spaces focuses on supporting queries that retrieve objects similar to a query point, when a metric distance function *dist* measures the objects' (dis)similarity. More formally, a metric space is a pair $M = (\Delta, dist)$, where Δ is a domain of feature values and *dist* is a distance function with the following properties:

1. $dist(p, q) > 0$, $q \neq p$ and $dist(p, p) = 0$ (non negativity),
2. $dist(p, q) = dist(q, p)$ (symmetry),
3. $dist(p, q) \leq dist(p, o) + dist(o, q)$ (triangle inequality).

The properties of the metric distance function express that a smaller distance between two objects means higher similarity. Therefore, the distance between identical objects should be zero, otherwise the distance has a positive value.

The similarity of two objects is symmetrical, and thus the distance function must also be symmetrical. The triangle inequality guarantees that the distance between two objects p and q is always smaller than the sum of the distances of p and q to any other object o.

Similarity search in metric spaces involves two different types of queries, namely *range* and *nearest neighbor* queries. Range queries are specified by a query object q and a range value r, and the answer set is defined to contain all the objects o from the dataset that have a distance to the query object q smaller than or equal to r:

Range Query $R(q, r)$**:** Given a query object q and a radius r, a point $p \in S$ belongs to result set R_q^r of the range query, if $dist(q, p) \leq r$.

A range query $R(q, r)$ can be interpreted as "retrieve all objects that are within distance r to q". The k-nearest neighbor (k-NN) query does not require a user to provide a radius for the query and is therefore easier to express than the similarity range query. The k-nearest neighbor query returns the k most similar data points from the dataset and is defined as follows:

k-nearest Neighbor Query $NN_k(q)$**:** Given a query object q and a positive integer k, the result set NN_q^k of the k-nearest neighbor is a set, such that $NN_q^k \subseteq S$, $|NN_q^k| = k$ and $\forall u, v : u \in NN_q^k, v \in S - NN_q^k$ it holds that $dist(q, u) \leq dist(q, v)$.

An example of a k-nearest neighbor query is "retrieve the k objects in S which are closest in distance to a given object". Given a query object q, a k-nearest neighbor query is equivalent to a range query specified by query point q and a radius equal to the distance of the k-th nearest neighbor. In this paper, we focus on range queries since k-NN queries can be transformed to range queries, if the distance of the k-th nearest neighbor is known. Radius estimation techniques for distributed nearest neighbor search have been studied in [14].

M-Tree. The M-Tree [9] is a distance-based indexing method, suitable for disk-based implementation. An M-Tree can be seen as a hierarchy of metric regions, also known as hyper-spheres or balls, as depicted in Fig. 1. More precisely, all the objects being indexed are referenced in the leaf nodes, while an entry in a non-leaf node stores a pointer to a node at the next lower level along with summary information about the objects in the subtree being pointed at. The objects in the internal nodes are database objects that are chosen (during the insertion) as representative points. For a non-leaf node N, the entries are quad-tuples $\{(p, r(p)), D, T\}$, where p is an representative object, $r(p)$ is the corresponding covering radius, D is a distance value, and T is a reference to a child node of N. The basic property is that for all objects o in the subtree rooted at T, we have $dist(p, o) \leq r(p)$. For each non-root node N, let object p' be the parent object, i.e. the object in the entry pointing to N. The distance value stored in D

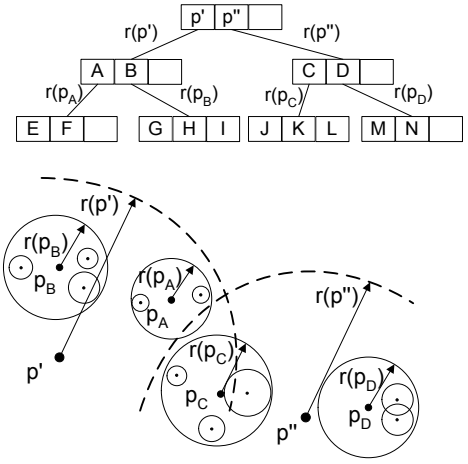


Fig. 1. Example of M-Tree

is the distance $dist(p, p')$ between p and the parent object p' of N. These parent distances allow more efficient pruning during search than would otherwise be possible. Similarly, for a leaf node N, the entries consist of pairs of the form (o, D), where o is a data object and D is the distance between o and the parent object of N.

The M-Tree is built in a bottom-up fashion and heuristics are presented in [9] for choosing the child node to insert an object and for splitting overflowing nodes. Bulk-loading strategies [8] have also been developed for use when an M-Tree must be built for an existing set of data objects. Moreover, the M-Tree is a dynamic index structure, therefore it can efficiently support updates. Range queries $R(q, r)$ for query object q and query radius r can be performed on the M-Tree using a depth-first search algorithm, initiated at the root. Entries of intermediate nodes N are pruned, when the query has no overlap with the ball represented by the the representative object p and its covering radius $r(p)$.

4 Routing Information for P2P Similarity Search

In this section, we first state the objectives that need to be attained by our framework to enable efficient similarity search. Then, we present the routing indices that are built at super-peer level. To elaborate on this, we first describe the local indices maintained at super-peer level, and then we introduce the metric-based routing indices.

4.1 Objectives for P2P Similarity Search

In applications like multimedia retrieval that require efficient similarity search, a server stores a collection of data objects, such as images, which refer to a

high-dimensional metric space and a distance function provides a measure of (dis)similarity. In order to support efficient similarity search over the stored data objects, an indexing technique, such as the M-Tree, is required. In our scenario, for distributed image similarity search, we assume that a set of servers, also mentioned as peers, store their data by using an M-Tree. Thus, each server is able to efficiently support similarity queries. The remaining challenge is to support efficient similarity search over the data stored at all peers in a distributed manner.

In this distributed architecture, the individual objectives that need to be attained for efficient processing of similarity queries are: 1) minimizing the number of required messages to retrieve the result, 2) minimizing the number of contacted peers (super-peers) that are involved in processing a similarity query, and 3) minimizing the maximum hop count for a given query, thereby minimizing the associated latency.

The first goal reflects the scalability of the system, in terms of consumed network resources. Obviously, the number of required messages should be kept small, in order to increase the number of queries that can be processed using the available bandwidth. The second goal relates to queries being processed only by peers and super-peers that actually store relevant results. Finally, the last goal is to achieve low latency for queries and minimize the total response time.

In order to achieve the aforementioned objectives, queries have to be forwarded during query processing only to peers and super-peers that may contribute to the query result set, while avoiding to contact peers and super-peers that store data that are not relevant to the query. Thus, instead of flooding queries at super-peer level, we build routing indices that describe the data that is available through each neighboring super-peer. Furthermore, each super-peer maintains information that summarizes the data available at each connected peer. Towards this goal, we use the hyper-spheres of the local M-Trees at each peer.

In the following, we first describe the local indexing at each super-peer (Section 4.2) which creates indices at each super-peers that can be used to determine which peers store relevant data. Then, we proceed by presenting our novel routing indices at each super-peer (Section 4.3) that summarize the data available through each neighboring super-peer.

4.2 Local Indexing

Each peer P_i that connects to a super-peer SP_j publishes a summary of its data, in order to make its content searchable by other peers. In our framework, we take advantage of the existing M-Tree index and each peer P_i publishes to its responsible super-peer SP_j, the hyper-spheres contained in the root of its M-Tree, as a summary of the stored data. This set of hyper-spheres covers all data objects stored at P_i, thus SP_j is able to determine if P_i stores data relevant to a potential range query, by searching for hyper-spheres that overlap with the query.

SP_j needs to support efficient retrieval of peer hyper-spheres, and consequently selection of the peers that store relevant data to a similarity query. For this purpose, SP_j inserts the collected hyper-spheres into a local M-Tree, also mentioned as *super-peer M-Tree*. This enables efficient similarity search over all data stored by peers associated to SP_j by contacting only the peers that store data that may appear in the query result set.

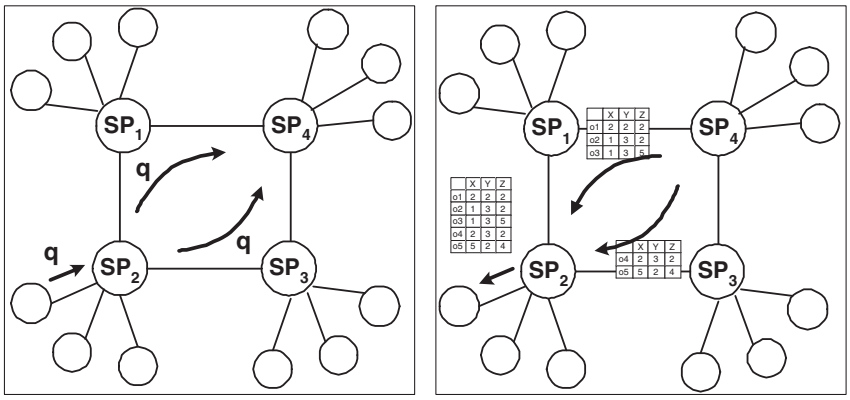
4.3 Routing Indices

The remaining challenge is to construct routing indices for processing similarity queries over the entire super-peer network. For this purpose, each super-peer maintains an M-Tree, also called *routing M-Tree*, to store hyper-spheres (collected from other super-peers) that describe the data accessible through each neighbor in the super-peer topology. The construction of routing indices at super-peer level is achieved in the following way.

A super-peer SP_i sends the descriptions of the hyper-spheres contained in the root of the super-peer M-Tree to its neighbors. This message has the following format: $(msgId, \{(p_m, r(p_m))\})$, where $msgId$ is an identifier that is unique for each SP_i, and $\{(p_m, r(p_m))\}$ represents the set of SP_i's hyper-spheres corresponding to the root of SP_i's M-Tree. Each hyper-sphere is defined by a representative object p_m and the corresponding covering radius $r(p_m)$. Each neighboring super-peer SP_j that receives a set of hyper-spheres for the first time performs two operations. First, SP_j stores locally the hyper-spheres in the routing M-Tree and attaches to them the identifier of the neighboring super-peer SP_i, from which the hyper-spheres were received. Second, SP_j propagates the hyper-spheres to all its neighbors, except for the one it received them from (SP_i). Any super-peer SP_k that is contacted by SP_j performs the same operations. However, notice that SP_k stores in its routing M-Tree the identifier of its neighbor SP_j together with the hyper-spheres, and not the identifier of the owner super-peer SP_i.

This construction protocol works also for network topologies that contain cycles. Since hyper-spheres of any super-peer SP_i are accompanied by a unique $msgId$, each recipient super-peer SP_k can perform duplicate elimination, in case SP_i's hyper-spheres are also received from a different network path. Notice that the granularity of the routing information stored at any super-peer is at the level of its neighbors $O(DEG_{sp})$ and not at the level of the network $O(N_{sp})$. Therefore, the constructed routing indices are scalable with network size.

We now elaborate more on the internal structure of nodes in the routing M-Tree. For internal nodes, the routing M-Tree entry is $\{(p, r(p)), D, T\}$, where $(p, r(p))$ is the representative object p and its covering radius $r(p)$, D is the distance to the parent object and T is a reference to a subtree. For leaf nodes, the routing M-Tree entry is $\{(p, r(p), SP(p)), D\}$, where $(p, r(p), SP(p))$ consists of the representative object p, its covering radius $r(p)$, and $SP(p)$ is the neighbor super-peer responsible for the hyper-sphere, whereas D is the distance to the parent object.



(a) Query propagation (b) Collecting results

Fig. 2. Super-peer query processing

5 Distributed Similarity Search

Our framework creates routing M-Trees on each super-peer and supports efficient query processing, in terms of local computation costs, communication costs and overall response time. In this section, we present our novel query routing algorithm that exploits the proposed routing indices.

A query may be posed by any peer P_q and is propagated to the associated super-peer SP_q, which becomes responsible for local query processing and query routing, and finally returns the result set to P_q. Each super-peer SP_i that receives the query performs two tasks: 1) SP_i routes the query to a subset of its neighboring super-peers, and 2) SP_i processes the query locally, in order to retrieve results from its local peers.

Afterwards, the relevant data is collected at SP_i and sent back to the neighboring super-peer from which the query was received. Finally, SP_q collects all results of its neighboring super-peers and sends the result set back to the peer P_q that posed the query. A high-level view of the query processing functionality is depicted in Fig. 2. The query propagation is shown in Fig. 2(a), starting from P_q and SP_q (in the example SP_2) to the rest of the super-peer network. Fig. 2(b) depicts the collection of query results back to SP_2.

We first elaborate on the details of local processing, and then the query routing mechanism is presented.

5.1 Super-Peer Local Query Processing

Given a range query $R(q, r)$, query processing at SP_i is performed by exploiting the summary information stored in the super-peer M-Tree. The aim is to retrieve the subset of local peers that need to be contacted. The peers that store data enclosed in the range query $R(q, r)$ have to be contacted, since these results are

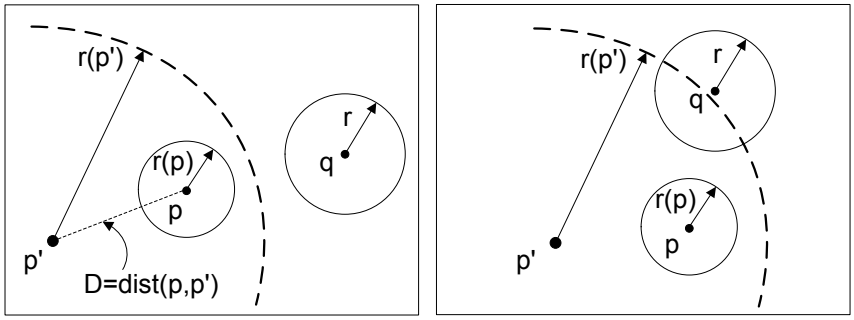
necessary to be retrieved and reported back to SP_q, in order to form the exact and complete result set. Therefore, SP_i uses its super-peer M-Tree to identify hyper-spheres of peers that intersect with the query. Recall that the retrieved hyper-spheres contain the peer identifier of the owner peer. Thus, the subset of peers that can contribute to the query result is determined and the range query is forwarded to the corresponding peers. This enables efficient similarity search over all data stored by peers associated to SP_i, since the query is posed only to peers having data that may appear in the result set, essentially forming an effective *peer selection mechanism* at super-peer SP_i. Each recipient peer processes the query using its local M-Tree, in the traditional way of processing range queries in M-Trees. Consequently, each peer reports its results to SP_i, which in turn is responsible for returning the results to SP_q.

5.2 Query Routing

After having described the local query processing on each super-peer, we proceed to present the details on query routing at super-peer level. Henceforth, we assume that each super-peer that receives the query also performs local query processing, as described above, therefore we omit the details of local query processing from the following discussion.

Given a range query $R(q, r)$, the querying super-peer SP_q needs to selectively propagate the query to a fraction of its neighboring super-peers as will be described shortly. Each intermediate super-peer SP_i that receives the query repeats the same process. The routing algorithm on any super-peer SP_i is based on its routing M-Tree. During query routing the summary information that is stored at the routing M-Tree of each super-peer SP_i is exploited. The aim of the routing algorithm is to retrieve all hyper-spheres that intersect with the query and the neighboring super-peers that should be contacted respectively. As a consequence, when a super-peer SP_r receives a range query $R(q, r)$, SP_r uses the routing M-Tree to efficiently retrieve all hyper-spheres that have an overlap with the query. Then, the set of neighbor super-peers is determined and the query is forwarded to them only. This forms the *super-peer selection mechanism* that enables routing of queries at super-peer level.

We now describe in more details how the set of relevant neighboring super-peers are retrieved from the M-Tree. Given a range query $R(q, r)$ defined by query object q and radius r, query routing is performed on the routing M-Tree by means of a depth-first traversal, initiated at the root. Let N be a node that is being visited with entry $\{(p, r(p)), D, T\}$, and let p' be its parent object. D represents the distance of p to its parent p', and T is a reference to a child node. The following two observations can be used for discarding subtrees during the M-Tree traversal. The first observation exploits the pre-computed distances $D{=}dist(p, p')$ to parent objects, in order to avoid computing the actual distance $d(p, q)$.



(a) Observation 1. (b) Observation 2.

Fig. 3. Conditions for pruning a subtree of routing M-Tree

Observation 1. Given a range query $R(q, r)$, if $|dist(p', q) - dist(p, p')| > r + r(p)$, then the subtree pointed by T can be safely pruned from the search.

The interpretation of the first observation is based on the fact that $|dist(p', q) - dist(p, p')| - r(p)$ is a lower bound of the distance of any object in the subtree pointed at by T. Thus, if the lower bound is greater than r, then no object in this subtree can be in the range (Fig. 3(a)). If the condition of Observation 1 is not satisfied, the distance $dist(p, q)$ must be computed. However, after having computed $dist(p, q)$, we can still avoid visiting the node pointed at by T, if the lower bound on the distance from q to any object p in T is greater than r. This is the case if $dist(p, q) > r + r(p)$, since the lower bound is $dist(p, q) - r(p)$ (Fig. 3(b)).

Observation 2. Given a range query $R(q, r)$, if $dist(p, q) > r + r(p)$, then the subtree pointed by T can be safely pruned from the search.

Capitalizing on these observations, Algorithm 1 describes the query routing process at a super-peer, in terms of determining the neighbor super-peers that need to be queried. It takes as input the range query $R(q, r)$ and the current tree node N. For each object p that belongs to node N (line 4), Observation 1 is used to prune the subtree T pointed by p (line 5). If T cannot be pruned, the distance $dist(p, q)$ is computed (line 6) and then the condition of Observation 2 is checked (line 7), in order to discard the subtree T pointed by p. If this condition does not hold, then we distinguish between two cases (line 8). If N is not a leaf node, then the algorithm is invoked on the subtree T (line 9). Otherwise, if N is a leaf node, then $SP(p)$ is added to the result (line 11). At the end of the algorithm, the list of neighboring super-peers to which the query should be routed is determined.

Algorithm 1. Query Routing QR

```
1: Input: N:node, q:query point, r:search radius
2: RES = ∅
3: let p' be the parent object of node N
4: for (∀p ∈ N) do
5:    if (|dist(p', q) − dist(p, p')| ≤ r + r(p)) then
6:       compute dist(p, q)
7:       if (dist(p, q) ≤ r + r(p)) then
8:          if (N is not a leaf) then
9:             QR(T, q, r)
10:         else
11:            RES = RES ∪ {SP(p)}
12:         end if
13:      end if
14:   end if
15: end for
16: return RES
```

6 Maintenance

In a dynamic P2P environment, maintenance of routing information is important, especially in the presence of data updates, peer joins and failures. In this section, we first discuss the maintenance cost of the routing indices. Afterwards, we describe how churn affects the proposed framework.

6.1 Data Updates

Maintenance of routing indices is triggered by data insertion, updates and deletions that occur at any peer. Any change of the data stored at a peer P_i causes updates to P_i's local M-Tree. However, as long as the peer's hyper-spheres contained in the root of the M-Tree do not change, such updates do not need to be propagated to any other super-peer.

In the case that a root hyper-sphere description changes, then P_i's responsible super-peer has to be informed. The super-peer updates its local M-Tree index and checks if the update changes the respective hyper-sphere descriptions at root level. Only if such a change occurs, need the other super-peers be updated. Otherwise, the change only affects the particular super-peer. Notice that if hyper-spheres shrink, then even if the super-peers are not updated immediately, the framework still provides correct answers to queries, at the cost of contacting more super-peers. In practice, a super-peer informs its neighbors about changes by broadcasting the modification in a similar way to the construction phase.

To summarize, data updates incur maintenance costs only if the hyper-spheres of a peer root, and eventually the super-peer's root hyper-sphere, are modified.

6.2 Churn

The existence of super-peers makes the system more resilient to failures compared to other P2P systems. Super-peers have stable roles, but in the rare case that a super-peer fails, its peers can detect this event and connect to another super-peer using the basic bootstrapping protocol.

On the other hand, a peer failure may cause the responsible super-peer to update its hyper-spheres. Only if churn rate is high, these changes need to be propagated to other super-peers. Even if updates are not propagated immediately after a peer fails, the only impact to our system is that the cost of searching is increased (i.e. super-peers no longer holding relevant results may be contacted), but the validity of the result is not compromised.

As already mentioned, a peer joins the network by contacting a super-peer using the bootstrapping protocol. The bootstrapping super-peer SP_B uses its routing clusters to find the most relevant super-peer to the joining peer. This is equivalent to the way similarity search is performed over the super-peer network. When the most relevant super-peer SP_r is discovered, the new peer joins SP_r. An interesting property of our approach is that joining peers become members of relevant super-peers, so it is expected as new peers join the system, that clustered datasets are gradually formed, with respect to the assigned super-peers.

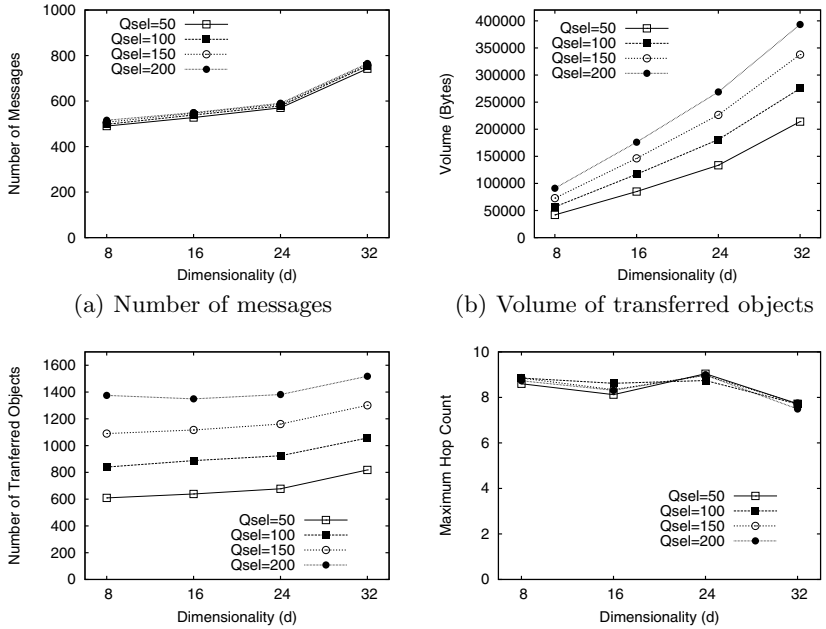
7 Experimental Evaluation

In order to evaluate the performance of our approach, we implemented a simulator prototype in Java. The simulations run on 3.8GHz Dual Core AMD processors with 2GB RAM. In order to be able to test the algorithms with realistic network sizes, we ran multiple instances of the peers on the same machine and simulated the network interconnection.

The P2P network topology used in the experiments consists of N_{sp} interconnected super-peers in a random graph topology. We used the GT-ITM topology generator[2] to create well-connected random graphs of N_{sp} peers with a user-specified average connectivity (DEG_{sp}). In our experiments we vary the network size (N_p) from 4000 to 20000 peers, while the number of super-peers varies from 200 to 1000. We also tested different DEG_{sp} values ranging from 4 to 7 and different number of peers per super-peer ($DEG_p = 20 - 100$). In addition, the query selectivity (Q_{sel}) of range queries is varied leading to queries that retrieve between 50 and 200 objects.

We used synthetic data collections, in order to study the scalability of our approach. Both uniform and clustered datasets are employed that were horizontally partitioned evenly among the peers. The uniform dataset includes random points in $[0, 10000]^d$. For the clustered dataset, each super-peer picks randomly a d-dimensional point and all associated peers obtain k_p cluster centroids that follow a Gaussian distribution on each axis with variance 0.05.

[2] Available at: `http://www.cc.gatech.edu/projects/gtitm/`



(a) Number of messages (b) Volume of transferred objects

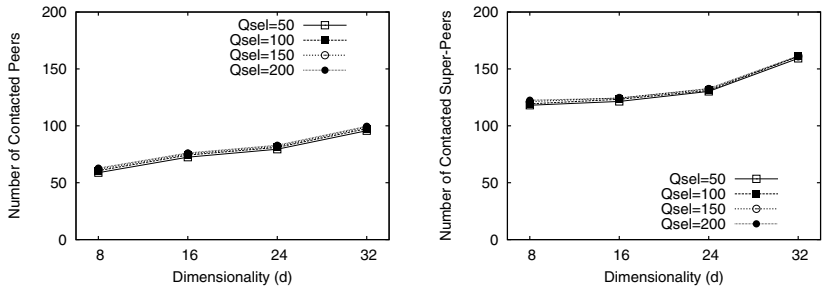
(c) Number of transferred objects (d) Maximum hop count

Fig. 4. Scalability with dimensionality for clustered dataset

Thereafter, the peers' objects are generated by following a Gaussian distribution on each axis with variance 0.025, and a mean equal to the corresponding coordinate of the centroid. We conduct experiments varying the dimensionality d (8-32d) and the cardinality n (4M-20M) of the dataset. We keep n/N_p=1000 in all setups. Additionally, we employed a real data collection (VEC), which consists of 1M 45-dimensional vectors of color image features, which was distributed to peers uniformly at random. In all cases, we generate 100 queries uniformly distributed and we show the average values. For each query a peer initiator is randomly selected. Although different metric distance functions can be supported, in this set of experiments we used the Euclidean distance function. We measure: (i) number of messages, (ii) volume of transferred data, (iii) number of transferred objects, (iv) maximum hop count, (v) number of contacted peers, (vi) number of contacted super-peers, and (vii) response time.

7.1 Scalability with Dimensionality

Initially, we focus on the case of clustered dataset. We use a default setup of: N_{sp}=200, N_p=4000, DEG_{sp}=4, n=4M, and the selectivity of range queries ranges from 50 to 200 objects. We study the effect of increasing dimensionality d to our approach. In Fig. 4(a), the number of messages required for searching increases when the dimensionality increases. The volume and the number of



(a) Contacted peers (b) Contacted super-peers

Fig. 5. Contacted peers and super-peers for clustered dataset

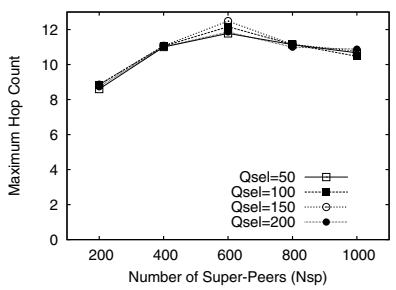
transferred objects are depicted in Fig. 4(b) and Fig. 4(c) respectively. Although the number of transferred objects is not significantly affected, the volume increases with dimensionality, as more bytes are necessary for representing objects of higher dimensionality. Notice that the transferred volume remains relatively low, between 100KB and 400KB. In Fig. 4(d), the maximum hop count is depicted, which relates to the latency of the approach. The number of required hops is between 8 and 9, irrespective of the increased dimensionality, which implies that latency is low, considering that the network consists of 4000 peers. Then, in Figs. 5(a) and 5(b), we measure the number of contacted peers and super-peers respectively. Although the number of super-peers that process the query is between 120 and 150, the number of peers is much lower, ranging from 60 to 100 peers.

7.2 Scalability with Network Parameters

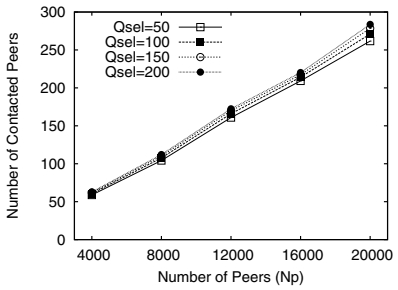
In the following, we study the scalability of our approach with respect to the network parameters, by fixing d=8. For this purpose, we increase the number of super-peers N_{sp} (Fig. 6(a)), peers N_p (Fig. 6(b)) and the average number of connections per super-peer DEG_{sp} (Fig. 6(c)). We observe that the maximum hop count increases only slightly, always remaining below 12, when the number of super-peers is increased by a factor of 5. On the other hand, in Fig. 6(b), the increasing number of peers only affects the number of contacted peers, however the increase is only marginal compared to the network size. Lastly, increasing the density of the super-peer network (DEG_{sp}=4-7) causes more messages to be transferred (500-700), in order to retrieve the result, but again the increase is small even in the case that the network becomes dense (DEG_{sp}=7).

7.3 Evaluation for Uniform Data

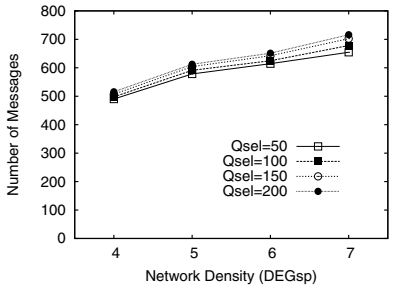
In Fig. 7, we examine the case of uniform data. Clearly, this is a hard case for our approach, as a query may in worst case have to contact all peers, in order to



(a) Scalability with N_{sp}



(b) Scalability with network size (N_p)



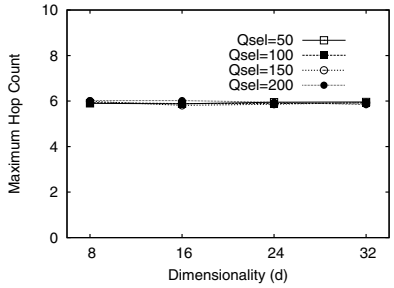
(c) Scalability with network density (DEG_{sp})

Fig. 6. Scalability for clustered dataset with respect to network parameters

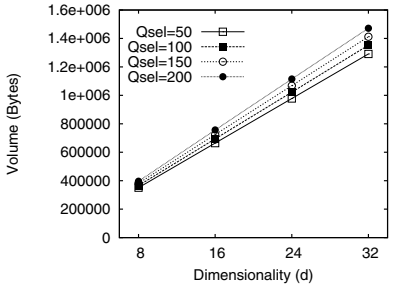
retrieve the correct results. This actually occurs in our experiments, causing also a large number of messages to be sent. However, notice that the maximum hop count, depicted in Figure 7(a), is small (around 6), even smaller than in the case of clustered dataset, since the probability of finding the results in smaller distance increases. We show the volume of transferred data in Fig. 7(b). Compared to the case of the clustered dataset, the total volume transferred increases by a factor of 3-4.

7.4 Evaluation for Real Data

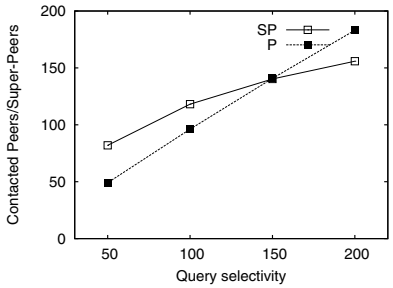
In addition, we evaluate our approach using a real dataset (VEC), which consists of 1M 45-dimensional vectors of color image features. We used a network of 200 super-peers and 1000 peers, thus each peer stores 1000 data points. In Fig. 7(c), the number of contacted peers and super-peers are depicted for increasing query selectivity from 50 to 200 points. The results are comparable to the case of the clustered dataset, but slightly worse, as the VEC dataset is not clustered. However, notice that the absolute numbers are comparable to the results obtained using the synthetic dataset, which is a strong argument in favor of the feasibility of our approach.



(a) Uniform data (maximum hop count)



(b) Uniform data (volume of transferred objects)



(c) Real data (contacted peers and super-peers)

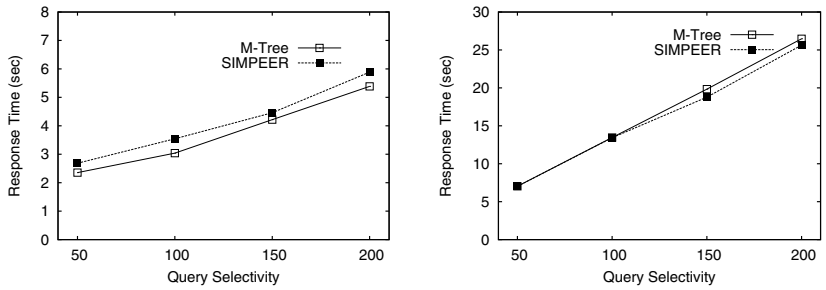
Fig. 7. Experiments with uniform and real datasets

7.5 Maintenance

We performed a series of experiments, in order to study the effect of updates on our system. For this purpose, we updated up to 100% of each peer's data, by deletions of points followed by insertions of new data points that follow the data distribution. Then, we measured the percentage of updates that actually triggered an update of a) the peer's hyper-spheres in the root of the peer M-Tree, and b) the super-peer's hyper-spheres in the root of the super-peer M-Tree. The first is a metric of the update costs from any peer to its super-peer, whereas the second indicates the update cost at super-peer level. The results show that only fewer than 2% of the updates lead to updates from peers to super-peers. More importantly, the percentage of updates that cause an update at super-peer level is only 0.1%. This shows that only a small percentage of data updates have an effect on the super-peer network, and such updates can still be efficiently managed using the protocols described in Section 6.

7.6 Comparison to SIMPEER

In Fig. 8, we study the comparative performance of the proposed framework to SIMPEER [14]. SIMPEER relies on the iDistance indexing technique instead of



(a) Uniform data (b) Clustered data

Fig. 8. Comparison to SIMPEER in terms of response time

the M-Tree. We performed a set of experiments using both approaches, assuming a modest 4KB/sec as network transfer rate, and we discuss our main findings here. We employ both a uniform and a clustered dataset, in order to explore the performance of both approaches. In the case of the uniform dataset, our framework outperforms SIMPEER in terms of response time, as depicted in Fig. 8(a). In contrast, when a clustered dataset is used, SIMPEER is marginally better than our framework, as shown in Fig. 8(b). The key factor that determines the individual performance achieved is the routing ability of the set of clusters that iDistance uses to summarize the data, compared to the hyper-spheres of the root of the M-Tree. For the clustered dataset, SIMPEER is able to accurately discover the underlying clusters in the data, resulting in better performance. When the data distribution is uniform, our framework based on M-Trees is more efficient than SIMPEER, since the performance of our metric-based routing indices is not influenced by the absence of a clustering structure in the data.

The M-Tree approach builds the index in a bottom-up manner and the insertion method as well as the block size influences the quality (and the number of) hyper-spheres of the root. On the other hand, iDistance relies on clustering and the employed clustering method influences its overall performance. A generic clustering algorithm, such as k-means, may lead to a poor performance, while an application-specific clustering method may improve the performance of routing. The advantage of our framework is that it does not require the existence of a clustering structure in the data.

8 Conclusions

Similarity search in metric spaces has several applications, such as image retrieval. In such applications that require similarity search in metric spaces, usually a server indexes its data with a state-of-the-art centralized metric indexing technique, such as the M-Tree. In this paper, we study the challenging problem of supporting efficient similarity queries over distributed data in a P2P system. We assume that each peer autonomously maintains its own data indexed

by an M-Tree. We propose a framework for distributed similarity search that exploits the local M-Trees, by using the hyper-spheres stored at the M-Tree roots as a summarization of the data. Based on this information, an efficient routing mechanism for similarity queries is built. The experimental results show that our approach performs efficiently in all cases, while the performance of our framework scales with all network and dataset parameters.

References

1. Banaei-Kashani, F., Shahabi, C.: SWAM: a family of access methods for similarity-search in peer-to-peer data networks. In: Proceedings of CIKM 2004, pp. 304–313 (2004)
2. Batko, M., Falchi, F., Lucchese, C., Novak, D., Perego, R., Rabitti, F., Sedmidubský, J., Zezula, P.: Building a web-scale image similarity search system. Multimedia Tools Appl. 47(3), 599–629 (2010)
3. Batko, M., Gennaro, C., Zezula, P.: A Scalable Nearest Neighbor Search in P2P Systems. In: Ng, W.S., Ooi, B.-C., Ouksel, A.M., Sartori, C. (eds.) DBISP2P 2004. LNCS, vol. 3367, pp. 79–92. Springer, Heidelberg (2005)
4. Batko, M., Novak, D., Falchi, F., Zezula, P.: On scalability of the similarity search in the world of peers. In: Proceedings of International Conference on Scalable Information Systems (InfoScale), vol. 20 (2006)
5. Bawa, M., Condie, T., Ganesan, P.: LSH forest: self-tuning indexes for similarity search. In: Proceedings of WWW 2005, pp. 651–660 (2005)
6. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: supporting scalable multi-attribute range queries. In: Proceedings of SIGCOMM 2004, pp. 353–366 (2004)
7. Chavez, E., Navarro, G., Baeza-Yates, R., Marroquin, J.L.: Searching in metric spaces. ACM Computing Surveys (CSUR) 33(3), 273–321 (2001)
8. Ciaccia, P., Patella, M.: Bulk loading the M-tree. In: Proceedings of Australasian Database Conference (ADC), pp. 15–26 (1998)
9. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 426–435 (1997)
10. Crainiceanu, A., Linga, P., Gehrke, J., Shanmugasundaram, J.: P-tree: a P2P index for resource discovery applications. In: Proceedings of WWW 2004 (2004)
11. Crainiceanu, A., Linga, P., Machanavajjhala, A., Gehrke, J., Shanmugasundaram, J.: P-ring: An efficient and robust p2p range index structure. In: Proceedings of SIGMOD, pp. 223–234 (2007)
12. Datta, A., Hauswirth, M., John, R., Schmidt, R., Aberer, K.: Range queries in trie-structured overlays. In: Proceedings of P2P 2005, pp. 57–66 (2005)
13. Dohnal, V., Sedmidubsky, J., Zezula, P., Novak, D.: Similarity searching: Towards bulk-loading peer-to-peer networks. In: Proceedings of International Workshop on Similarity Search and Applications (SISAP), pp. 87–94 (2008)
14. Doulkeridis, C., Vlachou, A., Kotidis, Y., Vazirgiannis, M.: Peer-to-peer similarity search in metric spaces. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 986–997 (2007)
15. Doulkeridis, C., Vlachou, A., Kotidis, Y., Vazirgiannis, M.: Efficient range query processing in metric spaces over highly distributed data. Distributed and Parallel Databases 26(2-3), 155–180 (2009)

16. Doulkeridis, C., Vlachou, A., Nørvåg, K., Kotidis, Y., Vazirgiannis, M.: Efficient search based on content similarity over self-organizing p2p networks. Peer-to-Peer Networking and Applications 3(1), 67–79 (2010)
17. Falchi, F., Gennaro, C., Zezula, P.: A Content–Addressable Network for Similarity Search in Metric Spaces. In: Moro, G., et al. (eds.) DBISP2P 2005 and DBISP2P 2006. LNCS, vol. 4125, pp. 98–110. Springer, Heidelberg (2007)
18. Ganesan, P., Bawa, M., Garcia-Molina, H.: Online balancing of range-partitioned data with applications to peer-to-peer systems. In: Proceedings of VLDB 2004, pp. 444–455 (2004)
19. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces. ACM Transactions on Database Systems (TODS) 28(4), 517–580 (2003)
20. Jagadish, H.V., Ooi, B.C., Tan, K.-L., Yu, C., Zhang, R.: iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. ACM Transactions on Database Systems (TODS) 30(2), 364–397 (2005)
21. Jagadish, H.V., Ooi, B.C., Vu, Q.H.: Baton: a balanced tree structure for peer-to-peer networks. In: Proceedings of VLDB 2005, pp. 661–672 (2005)
22. Jagadish, H.V., Ooi, B.C., Vu, Q.H., Zhang, R., Zhou, A.: VBI-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In: Proceedings of ICDE 2006, vol. 34 (2006)
23. Kalnis, P., Ng, W.S., Ooi, B.C., Tan, K.-L.: Answering similarity queries in peer-to-peer networks. Inf. Syst. 31(1), 57–72 (2006)
24. Liu, B., Lee, W.-C., Lee, D.L.: Supporting complex multi-dimensional queries in P2P systems. In: Proceedings of ICDCS 2005, pp. 155–164 (2005)
25. Novak, D., Batko, M., Zezula, P.: Large-scale similarity data management with distributed metric index. In: Information Processing and Management (2011)
26. Novak, D., Zezula, P.: M-Chord: a scalable distributed similarity search structure. In: Proceedings of International Conference on Scalable Information Systems (InfoScale), vol. 19 (2006)
27. Ntarmos, N., Pitoura, T., Triantafillou, P.: Range Query Optimization Leveraging Peer Heterogeneity in DHT Data Networks. In: Moro, G., Bergamaschi, S., Joseph, S., Morin, J.-H., Ouksel, A.M. (eds.) DBISP2P 2005 and DBISP2P 2006. LNCS, vol. 4125, pp. 111–122. Springer, Heidelberg (2007)
28. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: Proceedings of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), pp. 161–172 (2001)
29. Shen, H.T., Shu, Y., Yu, B.: Efficient semantic-based content search in P2P network. IEEE Trans. Knowl. Data Eng. 16(7), 813–826 (2004)
30. Shu, Y., Ooi, B.C., Tan, K.-L., Zhou, A.: Supporting multi-dimensional range queries in peer-to-peer systems. In: Proceedings of P2P 2005, pp. 173–180 (2005)
31. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), pp. 149–160 (2001)
32. Vlachou, A., Doulkeridis, C., Kotidis, Y.: Peer-to-Peer Similarity Search Based on M-Tree Indexing. In: Kitagawa, H., Ishikawa, Y., Li, Q., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 5982, pp. 269–275. Springer, Heidelberg (2010)
33. Vlachou, A., Doulkeridis, C., Mavroeidis, D., Vazirgiannis, M.: Designing a Peer-to-Peer Architecture for Distributed Image Retrieval. In: Boujemaa, N., Detyniecki, M., Nürnberger, A. (eds.) AMR 2007. LNCS, vol. 4918, pp. 182–195. Springer, Heidelberg (2008)