Noname manuscript No.
(will be inserted by the editor)

Efficient Range Query Processing in Metric Spaces over Highly Distributed Data

Christos Doulkeridis · Akrivi Vlachou · Yannis Kotidis · Michalis Vazirgiannis

Received: date / Accepted: date

Abstract Similarity search in P2P systems has attracted a lot of attention recently and several important applications, like distributed image search, can profit from the proposed distributed algorithms. In this paper, we address the challenging problem of efficient processing of range queries in metric spaces, where data is horizontally distributed across a super-peer network. Our approach relies on *SIMPEER* [13], a framework that dynamically clusters peer data, in order to build distributed routing information at super-peer level. *SIMPEER* allows the evaluation of exact range and nearest neighbor queries in a distributed manner that reduces communication cost, network latency, bandwidth consumption and computational overhead at each individual peer. In this paper, we extend *SIMPEER* by focusing on efficient range query processing and providing recall-based guarantees for the quality of the result retrieved so far. This is especially useful for range queries that lead to result sets of high cardinality and incur a high processing cost, while the complete result set becomes overwhelming for the user. Our framework employs statis-

Christos Doulkeridis · Akrivi Vlachou · Yannis Kotidis · Michalis Vazirgiannis
Department of Informatics, Athens University of Economics and Business, Greece
E-mail: {cdoulk,avlachou,kotidis,mvazirg}@aueb.gr

tics for estimating an upper limit of the number of possible results for a range query and each super-peer may decide not to propagate further the query and reduce the scope of the search. We provide an experimental evaluation of our framework and show that our approach performs efficiently, even in the case of high degree of distribution.

Keywords Similarity search · peer-to-peer · range queries

1 Introduction

During the last decades, the vast number of independent data sources and the high rate of data generation make central assembly of data at a single location infeasible. As a consequence, data management and storage become increasingly distributed. Moreover, as the number of participating sources increases, traditional client-server solutions are prone to bottleneck risks and therefore do not scale. Peer-to-peer (P2P) systems emerge as a powerful model for searching huge amounts of data distributed over independent sources in a completely distributed and self-organizing way.

In real-life P2P application scenarios, there exist some peers (called super-peers) that have special roles, due to enhanced capabilities. Super-peer infrastructures [29,30] harness the merits of both centralized and distributed architectures. Super-peer networks tackle the scaling and "single-point-of-failure" problems of centralized approaches, while exploiting the advantages of the completely distributed approach, where each peer builds an index over its own data. Super-peers accept a limited number of connections from peers and become responsible for building and maintaining a summary index over their peers' data. In addition, each super-peer maintains information about neighboring super-peers in the network (for example following the notion of routing indices [11]) for routing queries to remote peers.

Similarity search in metric spaces has received considerable attention in the database research community [6,16,21]. The objective is to find all objects that are similar to a given query object, such as a digital image, a text document or a DNA sequence. Numerous interesting applications can be deployed over a super-peer infrastructure that supports similarity search in metric spaces. In particular, distributed image retrieval, document retrieval in digital libraries, distributed search engines (e.g. for multimedia content), file sharing, as well as distributed scientific databases, are all examples of applications that can be realized over the proposed framework.

In this paper, we focus on the challenging problem of efficient similarity query processing for metric spaces in highly distributed P2P systems. Our approach, called *SIMPEER*, relies on a super-peer infrastructure and users who wish to participate, register their machines to the P2P system and each peer stores its own data. In order to make its data searchable by other peers, each peer autonomously clusters its data using a standard clustering algorithm, and sends the cluster descriptions as data summarization to its super-peer. Each super-peer maintains the cluster descriptions of its associated peers. Furthermore, to keep the information in a manageable size, each super-peer applies a clustering algorithm on the cluster descriptions of its peers, which results in a new set of cluster descriptions, also referred to as *hyper-clusters*, which summarize the data objects of all peers connected to the super-peer. Thus, each super-peer stores an abstract description of the data stored by its adjacent peers.

The remaining challenge is to answer queries over the entire super-peer network. Instead of flooding queries at super-peer level, we build routing indices based on the hyper-cluster descriptions that enable selective query routing only to super-peers that may actually be responsible of peers with relevant results. The routing index construction is based on communicating the hyper-cluster

descriptions that are further summarized into a set of *routing clusters*. This leads to a three-level clustering scheme, which summarizes the data stored at the peers at different level of detail, and enables efficient query processing, in terms of local computation costs, communication costs and overall response time for both range and k-NN queries. During query processing, each super-peer that receives the query, first uses its routing clusters to forward the query to those neighboring super-peers, which either have local results or some of their neighbors store query results. Then, the super-peer forwards the query only to the subset of its peers that hold data that may appear in the result set. Finally, each super-peer collects the results of its associated peers and the queried neighboring super-peers and sends the result set back to the querying peer. *SIMPEER* always retrieves the exact result set and by exploiting the routing indices, super-peers that can not contribute any results are not contacted during query processing.

Even though *SIMPEER* supports both range and k-nearest neighbor queries[1], in this paper we focus on range queries and deal with an intrinsic problem of range queries, namely the varying cardinality of the result. One limitation of range queries, in contrast to nearest neighbor queries, is that the cardinality of the result set is not known in advance, but can be anything between zero and the size of the data in the whole network. Consequently, the choice of an inappropriate value for the query range often leads to very few or too many query results. In the first case, a new range query has to be posed with a larger range, which leads to redundant processing cost. In the second case, more than necessary objects are retrieved, which again leads to increased processing cost. In our case of distributed query processing, the high cardinality of the result

[1] Nearest neighbor queries are processed by mapping them into range queries by employing distributed radius estimation techniques [13].

set incurs high processing and transfer costs, while contacting a large number of super-peers.

In order to alleviate this problem, in our proposed framework, we enrich the cluster information with statistical information that allows us to calculate the quality of the already retrieved result set, in terms of recall. Then, each super-peer is able to estimate, the potential number of retrievable results compared to the already retrieved answers. This enables recall-based query routing, since the query can be forwarded to some neighbors only, and still guarantee sufficient recall values. Our approach is applicable for several application scenarios. The simplest scenario is that the user sets a threshold in advance, which requires that only $x\%$ of the total results for a given query are enough, in order to reduce the response time. More interesting is the case that any super-peer may decide that the query result set is too large to be useful for the user, and bounds the search by a threshold, thus reducing the query processing cost through partial query evaluation. Each super-peer may take into consideration its workload and the availability in resources and define an appropriate threshold. Notice that our approach is easily extensible to progressive range query evaluation, since each super-peer can keep track of the evaluated queries and the paths pruned by the threshold. Then, each super-peer is able to deliver only the additional data objects, if the objects retrieved so far are not sufficient for the user.

The key contributions of our work are:

- We present similarity search algorithms for metric spaces, suitable for unstructured P2P settings, where peers are organized in a super-peer architecture.
- Our framework is based on a novel three-level clustering scheme that is maintained by super-peers, in order to avoid processing on peers and super-

peers that can not contribute to the result set. We introduce several pruning techniques that can speed up the evaluation of queries.
- To avoid huge and overwhelming result sets of range queries, we introduce statistics attached to the cluster information, that allow any super-peer to estimate the result set size and the already achieved result quality.
- We demonstrate how recall-based query routing is facilitated, while only data objects that belong to the query result set are retrieved. Our partial result set has guaranteed recall value. Furthermore, we effectively handle cycles in the network during query routing, as cycles affect the estimation of recall.
- We discuss the maintenance and construction cost of our framework and demonstrate that our approach provides a feasible solution for P2P similarity search.

Section 2 provides an overview of related work and in Section 3 we present the system overview. In Section 4, we present the necessary construction phase, while the actual query processing is described in Section 5. Thereafter in Section 6, our technique for recall-based range queries is introduced. In Section 7 the maintenance of routing indices is discussed. The experimental results are presented in Section 8, and finally we conclude in Section 9.

2 Related Work

The design of efficient indexing and query processing techniques over high-dimensional datasets has become an important research area, because of several database applications such as multimedia retrieval and time series matching that deal with high-dimensional data representations. In such applications, an important issue is to retrieve similar objects to a given query object. Typical

operations for similarity search include range and k-nearest neighbor (k-NN) queries.

Most algorithms assume that data objects are represented in a d-dimensional Euclidean space. Nevertheless, centralized indexing approaches [7,31,17] have also been proposed to handle efficient similarity search in metric spaces. A dynamic balanced index structure, called M-tree, for similarity search in metric spaces, was presented in [7]. A branch-and-bound technique is proposed to efficiently retrieve the k-nearest neighbor data objects. In [8], a cost model for querying the M-tree is developed that uses the distance distribution of objects. iDistance [31,17] is a state-of-the-art indexing method for similarity search that transforms the problem of similarity search to an one-dimensional interval search problem. For a survey of similarity query processing in metric spaces see [6,16].

Similarity search in P2P systems has attracted a lot of attention recently, however most existing approaches focus mainly on structured P2P systems or on building an overlay network that groups together peers with similar content. Recently, MCAN [14] and M-Chord [23] were proposed to handle similarity search in metric spaces, employing a structured P2P network. Both approaches 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 [17] to map objects into one-dimensional values. Afterwards, both approaches distribute the mapped objects over an underlying structured P2P system, namely CAN [25] and Chord [28] respectively. Queries are transformed into a series of interval queries, executed on existing distributed structured P2P networks. It is worth noticing that data preprocessing (clustering and mapping) is done in a centralized fashion, and only then data is assigned to peers. Relevant to this

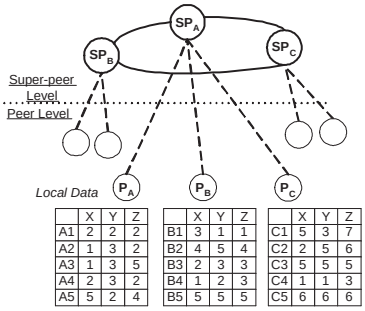
work, Batko et al. [3] present a comparative experimental evaluation of four distributed similarity search techniques (VPT*, GHT*, MCAN, M-Chord). VPT* and GHT* [2] are two distributed metric index structures where the dataset is distributed among peers participating in the network.

Recent works for similarity search in P2P systems focus on building a *suitable overlay topology*. A general solution for P2P similarity search for vector data is proposed in [1], named SWAM. Unlike structured P2P systems, 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. A P2P framework for multi-dimensional indexing based on a tree-structured overlay is proposed in [19]. LSH forest [4] 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 [26], 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 load-balancing system for range queries that extends Skip Graphs is presented in [27]. The use of Skip Graphs for range query processing has also been proposed in [15]. Several P2P range index structures have been proposed, such as Mercury [5], P-tree [9], BATON [18]. A variant of structured P2P for range queries that aims at exploiting peer heterogeneity is presented in [24]. In [22], 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 [20]. Their approach relies on a freezing technique, i.e. some queries are paused and can be answered by streaming results of other queries. Re-

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
k_p	Peer clusters
k_{sp}	Super-peer clusters
LC_p=$\{C_i : (K_i, r_i)\}$	List of peer clusters
LHC=$\{HC_i : (O_i, r_i')\}$	List of hyper-clusters
LRC=$\{RC_i : (R_i, r_i'')\}$	List of routing clusters

Fig. 1 Overview of symbols.



	X	Y	Z
A1	2	2	2
A2	1	3	2
A3	1	3	5
A4	2	3	2
A5	5	2	4

	X	Y	Z
B1	3	1	1
B2	4	5	4
B3	2	3	3
B4	1	2	3
B5	5	5	5

	X	Y	Z
C1	5	3	7
C2	2	5	6
C3	5	5	5
C4	1	1	3
C5	6	6	6

Fig. 2 Super-peer architecture

cently, in [10], P-Ring is proposed as an indexing structure that enables range query processing.

3 System Overview

In this paper, we assume a P2P system that relies on a super-peer infrastructure and users who wish to participate, register their machines (peers) to the P2P system. More formally, we assume a network of N_p peers, where each peer P_i holds n_i d-dimensional points, denoted as a set S_i $(i = 1..N_p)$. Obviously 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: $S = \cup S_i$. Given a space D defined by a set of d dimensions $\{d_1, d_2, .., d_d\}$ and a dataset S on D, a point $p \in S$ can be represented as $p = \{p_1, p_2, ..., p_d\}$ where p_i, is a value on dimension d_i. For a complete reference to the symbols used in this paper see the table depicted in Figure 1.

Some peers have special roles (Figure 2), due to their enhanced features, such as availability, 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 network, i.e. $N_{sp} << N_p$. Super-peers accept a maximum number of DEG_p connections from peers that join the network and directly

connect to one of the super-peers, and become responsible for building and maintaining a summary index over their peers' data. In addition, a super-peer is connected to a limited set of at most DEG_{sp} neighboring super-peers ($DEG_{sp} < DEG_p$) and maintains information about data available in the network through its neighbors for routing queries to remote peers.

Each peer maintains its own data objects, such as images or documents, which refer to a high-dimensional metric space and a distance function provides a measure of (dis)similarity. Similarity search in metric spaces focuses on supporting queries, whose purpose is to retrieve objects which are similar to a query point, when a metric distance function $dist$ measures the objects (dis)similarity. More formally, a metric space is a pair $M = (D, dist)$, where D is a domain of feature values and $dist$ is a distance function with the following properties: 1) $dist(p, q) = dist(q, p)$ (symmetry), 2) $dist(p, q) > 0$, $q \neq p$ and $dist(p, p) = 0$ (non negativity), 3) $dist(p, q) \leq dist(p, o) + dist(o, q)$ (triangle inequality). There are two types of similarity queries:

range query $R(q, r)$: Retrieve all elements that are within distance r to q, i.e. retrieve the result set $A \subseteq S$ such that $u \in A$ and $dist(q, u) \leq r$.

k-nearest neighbor query $NN_k(q)$: Retrieve the k closest elements to q, i.e. retrieve a set $A \subseteq S$ such that $|A| = k$ and $\forall u \in A, v \in S - A$, $dist(q, u) \leq dist(q, v)$.

Our approach enables similarity search in metric spaces over data distributed in a super-peer network, utilizing routing indices based on cluster information. Our framework is based on a novel three-level clustering scheme as described in the following.

4 Construction Phase

SIMPEER enables efficient query processing by using distributed routing information and guarantees that all objects that belong to the query result set are retrieved, without necessarily flooding the network. For this purpose, there exists a pre-processing phase, in which the routing information is exchanged and appropriate routing indices are built, that in turn allow subsequent query processing over the entire network of peers. Our approach relies on and extends iDistance [31,17], a centralized state-of-the-art approach that takes advantage of cluster information, in order to effectively maintain high-dimensional data objects and support similarity search in metric spaces. *SIMPEER* uses clustering information as a multi-dimensional summary of all data objects stored at a peer and utilizes a three-level clustering scheme:

- Each peer clusters its own data and the resulting clusters are used to index local points using iDistance.
- A super-peer receives cluster descriptions from its peers and computes the hyper-clusters using our extension of iDistance. Hyper-clusters are used by a super-peer to decide which of its peers should process a query, essentially forming a peer selection mechanism.
- Hyper-clusters are communicated among super-peers and are further summarized, in order to build a set of routing clusters. These are maintained at super-peer level and they are used for routing a query across the super-peer network, thus creating a super-peer selection mechanism.

In this section, we discuss the pre-processing phase in detail.

4.1 Peer Construction Phase

Each peer is responsible for its own data, which is organized and stored based on iDistance [31,17]. First, the peer applies a clustering algorithm on its local data. Even though the choice of the algorithm influences the overall performance of the system, each peer may choose any clustering algorithm. The clustering algorithm leads to a set of k_p clusters LC_p=$\{C_i : (K_i, r_i)|1 \leq i \leq k_p\}$. Each cluster is described by a cluster centroid K_i (also mentioned as reference object) and a radius r_i, which is the distance of the farthest point of the cluster to K_i. Each data object is assigned to the nearest cluster C_i, based on its distance to K_i.

Assuming a data space partitioning into k_p clusters, each data object is assigned a one-dimensional iDistance value according to the distance to its cluster's reference object. Having a constant c to separate individual clusters, the iDistance value for an object $x \in C_i$ is

$$iDist(x) = i * c + dist(K_i, x)$$

Expecting that c is large enough, all objects of the i-th cluster are mapped to the interval $[i * c, (i + 1) * c]$, as shown in Figure 3.

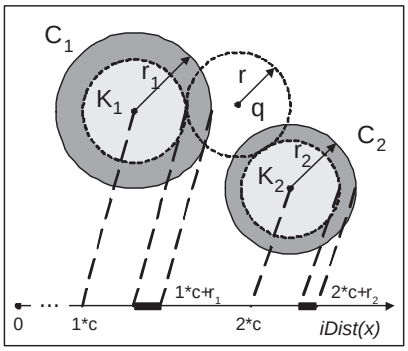


Fig. 3 *iDistance* mapping to 1-dimensional values.

The actual data objects are stored in a B^+-tree using the iDistance values as keys. Additionally, the list of the clusters LC_p, i.e. the cluster centroids K_i and the cluster radii r_i are kept in a main memory list. Thus, the problem of similarity search is transformed to an interval search problem, as will be described in Section 5.1.

4.2 Super-peer Construction Phase

Each super-peer SP_A collects the cluster descriptions from its associated peers, i.e. the lists $\{LC_{p_i}|1 \leq i \leq DEG_p\}$. It should be emphasized that only the cluster descriptions as a summarization of the peer's data is published to the super-peer, while the original data is stored by the peer itself. In order to keep the information in a manageable size, SP_A applies a clustering algorithm on the cluster descriptions of its peers, which results in a new set of cluster descriptions, also referred to as a list of *hyper-clusters*, which summarize the data objects of all peers connected to the super-peer. The peer cluster descriptions are mapped to one-dimensional values, using our extension of iDistance, in such a way that a range query can be mapped into an one-dimensional interval search. In the following, we present our extension of iDistance that facilitates one-dimensional mapping of clusters.

The peer cluster descriptions collected at super-peer SP_A are represented by a list LC_{sp}=$\{(K_1, r_1), ..., (K_{n_{sp}}, r_{n_{sp}})\}$, where n_{sp} is the total number of clusters of SP_A's peers. For the sake of simplicity, we assume that n_{sp}=$k_p * DEG_p$, i.e. all peers have the same number of clusters k_p. Following the iDistance concept, SP_A applies a clustering algorithm on the list LC_{sp} which results in a list of hyper-clusters LHC_{sp}=$\{HC_i : (O_i, r'_i)|1 \leq i \leq k_{sp}\}$, where k_{sp} the number of hyper-clusters, O_i the hyper-cluster centroid and r'_i the

hyper-cluster radius, which is the distance of the farthest point of all clusters assigned to the hyper-cluster to O_i.

Each cluster C_j is mapped to a one-dimensional value based on the nearest hyper-cluster centroid O_i using formula:

$$key_j{=}i * c + [dist(O_i, K_j) + r_j]$$

which practically maps the farthest point of a cluster C_j based on the nearest reference point O_i. Similarly to iDistance, the one-dimensional values are indexed using a B^+-tree. The B^+-tree entry e_j consists of the cluster's center K_j, its radius r_j and the distance d_j to its nearest reference point:

$$e_j : (key_j, K_j, r_j, d_j, IP_j)$$

Additionally, in the B^+-tree entry, the IP address of the peer is stored, in order to be able to propagate the query to those peers that have clusters that intersect with the query.

Furthermore, for each hyper-cluster HC_i, except from the radius r'_i, we also keep a lower bound ($dist\_min_i$) of all cluster distances. The distance $dist\_min_i$ is the distance of the nearest point of all clusters C_j to O_i. These two numbers practically define the effective data region of reference point O_i, or in other words, the region in HC_i where all points of all clusters C_j belong to.

4.3 Routing Indices Construction

Each super-peer builds a variant of routing indices, in order to efficiently route queries to the appropriate neighboring super-peers. The routing information consists of assembled hyper-clusters HC_i of other super-peers. Thus, for each neighboring super-peer a list of hyper-clusters is maintained, corresponding to hyper-clusters that are reachable through this particular neighboring super-peer. During query routing, the routing indices are used to prune neighboring

super-peers, thus inducing query processing only on those super-peers that can contribute to the final result set.

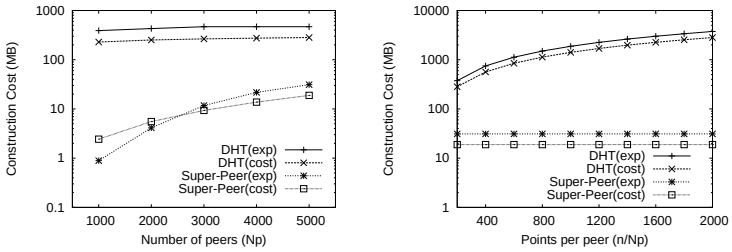
In more detail, each super-peer SP_A broadcasts its hyper-clusters using *create* messages in the super-peer network. Then, each recipient super-peer SP_r reached by *create* messages, assembles the hyper-clusters of other super-peers and uses the extension of the iDistance method for storing this information. Therefore, SP_r runs a clustering algorithm on the assembled hyper-clusters, which results in a set of *routing clusters (RC)* that constitute a summary of the hyper-cluster information. Then SP_r indexes the hyper-clusters, in a completely analogous manner as it clustered its peers clusters into hyper-clusters. The only difference is that for each routing cluster, the identifier of the neighboring super-peer, from which the owner of the hyper-cluster is accessible, is additionally stored.

Summarizing, a super-peer SP_A uses our extension of iDistance in two ways. First, it clusters its peers' clusters $\{LC_{p_i}|1 \leq i \leq DEG_p\}$ into hyper-clusters HC_i and indexes this information in a B^+-tree. SP_A also clusters the hyper-clusters $\{LHC_i|1 \leq i \leq N_{sp} - 1\}$ collected from other super-peers, resulting in a list of routing clusters LRC_A. These are then used to index LHC_i in a separate B^+-tree using the one-dimensional mapping, analogously to the technique employed for its peers' clusters.

4.4 Construction Cost

In this section, we compare by means of an analytical model the construction cost of *SIMPEER* to the construction cost of a DHT-based approach (e.g. similar to M-Chord [23]).

The cost of indexing one data point in a DHT is logarithmic with respect to the network size (N_p). Let d be the dimensionality of the data and b the size



(a) Increasing number of peers (b) Increasing number of points/peer

Fig. 4 Construction cost of a DHT-based versus a super-peer approach

of each variable (for example double), then the cost for publishing one point is: $d \times b \times \log N_p$. If n/N_p denotes the number of points per peer, the cost incurred by a peer P_i publishing its data is: $C_i = n/N_p \times d \times b \times \log N_p$. Thus the total cost for all peers is: $COST_{DHT} = \sum_{i=1..N_p} C_i = n \times d \times b \times \log N_p$.

For the super-peer approach, we denote k_p the number of clusters per peer, k_{sp} the number of clusters per super-peer, and N_{sp} the number of super-peers. The cost of N_p peers publishing their k_p clusters (centroid and radius) to their super-peers is: $C_{peer} = N_p \times k_p \times b \times (d+1)$. Assuming an acyclic super-peer network, the cost of broadcasting some information at super-peer level is $O(N_{sp})$ and the cost of super-peer hyper-cluster exchange is: $C_{super-peer} = N_{sp}^2 \times k_{sp} \times b \times (d+1)$. Then the total cost is: $COST_{sp} = b \times (d+1) \times [N_p \times k_p + N_{sp}^2 \times k_{sp}]$.

In Figure 4, we show some charts that depict graphically the results of this simple cost analysis. In addition, in order to validate our cost model, we show the results produced by 1) our framework experimentally, denoted as *Super-Peer(exp)*, and 2) a simulator of Chord we developed, denoted as *DHT(exp)*. The following default values are used: d=10, b=8, k_p=20, k_{sp}=10. In Figure 4(a), we study the effect of increasing network size. This is for $n = 1M$ points and using DEG_p=50 peers per super-peer. The super-peer approach is cheaper than the DHT approach for this setup, however the super-

peer approach is more sensitive to N_p. Moreover, the experimental results are similar to the results of the cost model. The small discrepancies observed are expected, as the super-peer topology is not acyclic and the actual cost of broadcasting depends on the density of the super-peer topology.

In the second chart (Figure 4(b)), we gradually increase the cardinality of the dataset from 1M to 10M, for N_p=5000 and DEG_p=50. It is obvious that the DHT approach is sensitive to n, while the cost of the super-peer approach does not depend on n at all. These results show that the construction cost of a super-peer based approach is comparable to a DHT-based approach, and often it is even cheaper depending on the parameters. Again, the experimental results closely follow the results of the cost model, verifying its correctness.

5 Distributed Query Processing

In super-peer architectures, queries are typically routed first in the super-peer backbone and afterwards, if necessary, they are distributed to the peers that are connected to the super-peers. An important parameter is the super-peer topology, which influences the performance of routing. In this work we assume that the pre-defined super-peer topology is generic and not restricted to a particular form, and we focus on the optimization of interactions among super-peers and peers.

Given a range query $R(q, r)$, each super-peer SP_A that receives the query uses its routing clusters to detect the hyper-clusters that intersect with the query and forwards the query to the corresponding neighboring super-peers. These hyper-clusters may summarize clusters that belong to peers associated to the neighboring super-peer or hyper-clusters of other super-peers that are accessible through the neighboring super-peer. Therefore the query is forwarded to SP_A's neighbors, which in turn propagate it to their neighbors.

Thereafter, SP_A forwards the range query only to those of its associated peers that have clusters intersecting with the query, or in other words to peers that hold data that may appear in the result set and should therefore be examined. Finally, SP_A collects the results of its associated peers and the queried neighboring super-peers and sends the result set back to the super-peer from which SP_A received the query.

In this section, we first focus on how query processing is performed by a single peer. Then, we present the query processing performed by a single super-peer, which consist of two algorithms: 1) the algorithm used by the super-peer to determine those of its associated peers that probably store data that belong to the result set, and 2) a routing algorithm that allows a super-peer to choose the subset of its neighbors that can lead to query results.

5.1 Peer Query Processing

When a peer receives a query, this means that probably this peer stores some data objects relevant to the query. Then, the peer is responsible to evaluate the query according to its own data. Since the data objects are organized and stored based on the iDistance concept, the peer exploits the existing index to efficiently evaluate the query.

For a given range query $R(q, r)$, the peer examines each cluster in the list LC_p and searches separately those clusters that possibly contain objects matching the query. Algorithm 1 describes how range query processing on a peer is performed. Practically, for each peer cluster $C_i \in LC_p$, the algorithm tests if the query intersects the cluster area (line 4). Thus, if a cluster C_i satisfies the inequality $dist(K_i, q) - r \leq r_i$, an interval search $[dist(K_i, q) + i * c - r, dist(K_i, q) + i * c + r]$ is posed on the B^+-tree. This iDistance interval corresponds to the area of C_i that should be scanned, in or-

Algorithm 1 Peer query processing.

```
1: Input: (q, r)
2: Output: Result set S
3: for C_i ∈ {LC_p} do
4:   if (d(K_i, q) − r ≤ r_i) then
5:     cursor ← B+tree_range_query[dist(K_i, q) + i ∗ c − r, dist(K_i, q) + i ∗
       c + r]
6:     while (candidate = has_next(cursor)) do
7:       if (dist(candidate, q) ≤ r) then
8:         S ← S ∪ {candidate}
9:       end if
10:    end while
11:  end if
12: end for
13: return S
```

der to find all relevant objects. After these objects are retrieved, a refinement step is required, due to the lossy mapping of iDistance, which maps different equidistant points from K_i to the same one-dimensional value. In order to ensure that the retrieved object is indeed within a distance r from the query, in the refinement step, each object's distance to q is computed and if it is smaller than r (line 7), the object is added to the result set S (line 8). For example in Figure 3 the range query intersects with both clusters C_1, C_2. According to the iDistance values, all objects falling in the dark grey shadowed area are retrieved and examined whether they belong to the result set.

5.2 Super-Peer Query Processing

When a super-peer receives a query, the super-peer first routes the query to those neighbors that may contribute to the query, and then it exploits the hyper-cluster information to detect its associated peers' clusters that intersect with the query. Then, the super-peer propagates the query to them. In this section, we first provide an algorithm that retrieves all peer clusters that in-

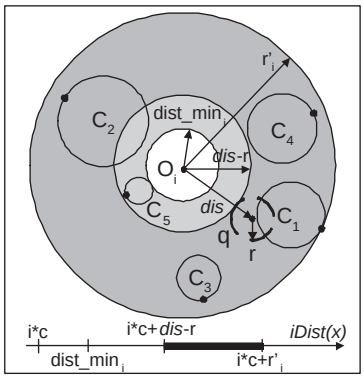


Fig. 5 Covering region for a hyper-cluster O_i, and search interval for a range query $R(q, r)$.

tersect with a given range query $R(q, r)$. Then, we describe how query routing is performed.

A super-peer needs to determine the clusters and, consequently, also the peers, that intersect with the range query, while the actual data is accessed directly from peers during query processing. Based on the one-dimensional mapping, in order to retrieve all clusters that belong to a hyper-cluster HC_i an interval search $[i * c + dist\_min_i,\ i * c + r'_i]$ on the iDistance values is posed, since the region $[dist\_min_i,\ r'_i]$ contains all clusters assigned to the hyper-cluster HC_i. In the following, we denote with dis the distance of O_i to q. The goal of our search algorithm is to filter out clusters that do not intersect with the query, based on the iDistance values. Since the clusters are mapped to one-dimensional values with respect to the farthest points of each cluster, searching all indexed points until r'_i cannot be avoided. This is clearly depicted in Figure 5 by means of an example. The hyper-cluster radius r'_i is defined by the farthest point of the cluster C_1, whereas $dist\_min_i$ is defined by cluster C_5. The query intersects with C_1 that is mapped to an iDistance value based on the r'_i distance. In other words, it is not enough to search until $dis + r$, since some farthest points of intersecting clusters may be overlooked.

Algorithm 2 Super-peer query processing.

```
1:  Input: (q, r)
2:  Output: Result set S
3:  for HC_i ∈ {LHC} do
4:     dis ← dist(O_i, q)
5:     lower ← max(dis − r, dist_min_i)
6:     if (dis − r ≤ r'_i) and (dis + r ≥ dist_min_i) then
7:        cursor ← B+tree_range_query[i ∗ c + lower, i ∗ c + r'_i]
8:        while (C_j : (K_j, r_j) = has_next(cursor)) do
9:           if (dist(K_j, q) ≤ r + r_j) then
10:             S ← S ∪ {C_j}
11:          end if
12:       end while
13:    end if
14: end for
15: return S
```

The starting point of the interval search is the iDistance value corresponding to $max(dis - r, dist\_min_i)$. For the query $R(q, r)$, in our example (Figure 5), the search scans the interval $[i * c + dis - r, i * c + r'_i]$.

Algorithm 2 describes the range query search algorithm performed by super-peer. Range query search takes as input a query point q and a radius r. The range search algorithm essentially consists of three steps: 1) it checks whether the hyper-cluster HC_i can provide relevant results (line 6), 2) (if so) it locates a starting point, denoted as $lower = max(dis - r, dist\_min_i)$ (line 5), for starting an interval search on the B^+-tree (line 7), and 3) scans the interval until r'_i (line 7). Since the one-dimensional mapping used by the extension of the iDistance is lossy, we have to remove the clusters that do not intersect with the query by a refinement step. In line 9, our algorithm tests if the cluster actually intersects with the given query.

As regards query routing, the routing indices present at any super-peer are used to prune neighboring super-peers, thus inducing query processing only on those super-peers that can contribute to the final result set. More formally, given a query $R(q, r)$ and a set of hyper-clusters HC_i:(O_i, r'_i), a neighboring

super-peer is pruned, if for all of its hyper-clusters HC_i it holds:

$$dist(O_i, q) > r + r'_i$$

Since the hyper-clusters are stored at each super-peer by using the extension of iDistance, in a similar way as the peers' clusters, the algorithm used to take the routing decision at a super-peer is completely analogous with Algorithm 2. Instead of examining the hyper-cluster, the super-peer examines the routing clusters and retrieves hyper-clusters from the B^+-tree, that belong to other super-peers, instead of clusters of the associated peers.

6 Lower Bound for Recall-based Range Queries

In this section, we show that by exploiting statistical information attached to the cluster descriptions, our approach can provide a lower bound for achieved recall of range queries. This is particularly useful for applications that partial results are adequate, as the initiator super-peer may set a threshold that requires in advance that for a given query only x% of the total results are enough, in order to reduce the search costs. The lower bound estimation can be performed at any intermediate super-peer during query routing. If the computed lower bound exceeds the given threshold, the search is interrupted and results can be returned to the user.

In the following, we present some definitions and then we discuss how to enrich cluster descriptions with extra information, in order to enable any super-peer to compute a guaranteed recall value at that point. Furthermore, we describe recall-based query routing assuming an acyclic super-peer topology and then we enhance our approach to handle cycles effectively.

6.1 Preliminaries

Recall is defined as the ratio of the number of relevant objects (A) retrieved to the total number of relevant objects (B) to the query. In order to provide a lower bound for recall, we need an upper bound for the total number of relevant data objects that may be retrieved (B). At any given point during routing, the super-peer knows how many objects have been retrieved so far (A), and can estimate an upper bound for B.

Each peer cluster C_i is augmented with the number of points $n(C_i)$ contained in C_i. Then a super-peer collects its peers' clusters and generates a set of hyper-clusters. For each hyper-cluster HC_i, the number of points belonging to HC_i can be easily computed as the sum of points in enclosed clusters $n(HC_i) = \sum n(C_i)$, and $n(HC_i)$ is attached to the hyper-cluster. This extra information is maintained with the cluster description and as will be shown in the following, it is sufficient for any super-peer that processes the query to compute a guaranteed recall value, based on the results retrieved thus far.

6.2 Determining the Lower Bound for Recall

Let us consider a range query $R(q, r)$ initiated at a super-peer SP_q. First, using Algorithm 2, SP_q determines the number of results to $R(q, r)$ that belong to SP_q's local peers, denoted as n_{local}. Then, SP_q uses its routing clusters, in order to identify other super-peers' hyper-clusters that intersect with the query. By adding the number of points in these hyper-clusters, SP_q determines an upper bound of points in range query $R(q, r)$ that can be retrieved from each of its super-peer neighbors. Therefore, each super-peer can provide a lower bound for recall based on its view of the rest of the network, captured in the hyper-cluster information of other super-peers.

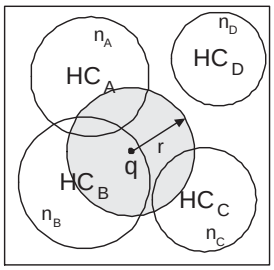


Fig. 6 A range query $R(q, r)$ that intersects with hyper-clusters HC_A, HC_B and HC_C.

In more detail, when a query intersects with a hyper-cluster, the number of points in the hyper-cluster is the maximum number of objects that may be retrieved. Obviously, fewer objects will be actually retrieved from this hyper-cluster, as the intersection area may not cover the entire hyper-cluster. Nevertheless, we are interested in computing an upper bound of retrievable objects, so in this way we can compute an upper bound n_i for each hyper-cluster HC_i that intersects with the query.

Example: Consider the case of a super-peer SP_q with four neighboring super-peers SP_A, SP_B, SP_C and SP_D. Let us further assume that SP_q initiates a range query $R(q, r)$ and after examining the hyper-clusters that SP_q has collected, SP_q determines that the range query overlaps with hyper-clusters HC_A, HC_B and HC_C that belong to super-peers SP_A, SP_B and SP_C respectively. This is graphically depicted in Figure 6, describing SP_q's viewpoint. Furthermore, SP_q determines an upper bound of the number of points that can be retrieved through each neighbor, for instance n_A results through SP_A. Now, if SP_q has retrieved n_{local} results to the query from its peers, then SP_q computes a lower bound for recall as: $\frac{n_{local}}{n_A+n_B+n_C}$. Notice that SP_D's hyper-cluster does not intersect with the query, hence SP_q determines that SP_D can not contribute to the query result.

Algorithm 3 Local bound computation at SP_q.

1: **Input:** (q, r)
2: **Output:** Lower bound for recall
3: $L_{nei}[] \leftarrow SP_q$.getNeighbors() // List of SP_q's neighbors
4: int $n[]$ // List of retrievable results per SP_q's neighbor
5: $n_{local} \leftarrow SP_q$.RangeQuery$(q, r)$ // Using Algorithm 2
6: **for** $RC_i \in \{LRC\}$ **do**
7: $dis \leftarrow dist(R_i, q)$
8: **if** $(dis - r \leq r_i'')$ **then**
9: **for** $HC_j \in RC_i$ **do**
10: $dis \leftarrow dist(O_i, q)$
11: **if** $(dis - r \leq r_i')$ **then**
12: **if** $(HC_j \in L_{nei}[k])$ **then**
13: $n[k] \leftarrow n[k] + n_j$
14: **end if**
15: **end if**
16: **end for**
17: **end if**
18: **end for**
19: **return** $(n_{local}, \sum_{i \in L_{nei}} n[i])$

Algorithm 3 describes how the lower bound for recall is determined at querying super-peer SP_q. For each routing cluster RC_i that intersects with the query (line 8), the algorithm examines which hyper-cluster HC_j that belongs to RC_i intersects with the query (line 11). Then the neighbor from which HC_j was collected at SP_q is determined (line 12), which is indexed in the k-th position in the list L_{nei} of neighboring super-peers. Then, the number of results that can be retrieved through this neighbor is increased by the number of points n_j in HC_j (line 13). Finally, the lower bound for recall can be computed as the fraction of local results (n_{local}) found at SP_q over the total number of retrievable results ($\sum_{i \in L_{nei}} n[i]$) through SP_q's neighbors.

6.3 Recall-based Query Routing

After having described how local range query processing is performed at a super-peer and the exact way to determine a lower bound for recall, we pro-

Algorithm 4 Lower bound computation at SP_i.

1: **Input:** $(q, r), n_{other}, n_{found}$
2: **Output:** Lower bound for recall LB
3: $(n_{local}, n_{nei}) \leftarrow SP_i$.LocalBoundComp$(q, r)$ // Using Algorithm 3
4: $LB = \frac{n_{local}+n_{found}}{(n_{local}+n_{found})+n_{other}+n_{nei}}$
5: **return** LB

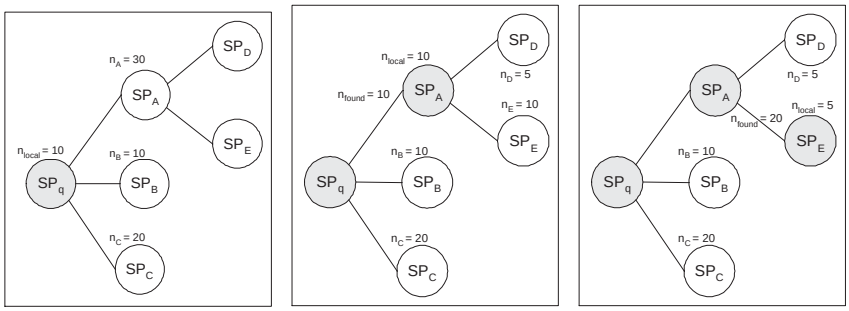
ceed to describe how query routing is performed. The basic idea is to employ a depth-first search algorithm that selects the most promising paths in the network. During query routing, any super-peer that receives the query chooses the neighbor that has the largest number of potential results. Intuitively, this strategy aims at contacting first those super-peers that actually maintain a large fraction of the requested results. In the following, we assume an acyclic super-peer topology.

In more detail, a super-peer SP_r that receives the query $R(q, r)$ from another super-peer SP_s, also receives from SP_s: a) the number of results (n_{found}) retrieved so far at previous super-peers, and b) the number of results that can potentially be retrieved by routing the query to other paths (n_{other}). Then, SP_r uses Algorithm 4 and computes its local results (n_{local}), and the number of results (n_{nei}) that can be retrieved from its neighbors. Thus, the lower bound for recall is computed as:

$$\frac{n_{found}+n_{local}}{(n_{found}+n_{local})+n_{other}+n_{nei}}$$

If the achieved recall is not sufficient, then SP_r decides to forward the query to that neighbor that can contribute most to the retrieved results (n_{nei}).

Example: Let us consider again SP_q that initiates a range query $R(q, r)$. Assume that SP_q has three neighbors, namely SP_A, SP_B and SP_C, as depicted in Figure 7(a). SP_q processes the range query locally and determines $n_{local} = 10$ results. SP_q also uses its routing clusters to retrieve hyper-clusters of other super-peers that intersect with the query. Thus, SP_q computes that $n_A = 30$ results can be potentially retrieved by following the path starting at SP_A,



(a) SP_q decides to forward the query to SP_A.

(b) SP_A computes the lower bound for recall.

(c) The query is forwarded to SP_E.

Fig. 7 Recall-based query routing for a range query $R(q, r)$ initiated at super-peer SP_q.

$n_B = 10$ through SP_B and $n_C = 20$ through SP_C. Therefore, SP_q routes the query to the most promising neighbor, aiming at maximizing the number of retrievable results. Any intermediate super-peer, say SP_A, that receives the range query $R(q, r)$, also receives from SP_q: a) the number of results (n_{found}) retrieved so far at previous super-peers (in this case $n_{found} = n_{local} = 10$), and b) the number of results that can potentially be retrieved by routing the query to other paths ($n_{other} = n_B + n_C = 30$). SP_A first computes the number of local results ($n_{local} = 10$) to the range query (see Figure 7(b)). SP_A can also compute an upper bound of the number of results to $R(q, r)$ (Algorithm 3) that can be retrieved through its neighbors SP_D ($n_D = 5$) and SP_E ($n_E = 10$). Thus, SP_A determines the lower bound for recall as: $\frac{n_{found}+n_{local}}{(n_{found}+n_{local})+(n_{other}+n_D+n_E)}$. In case this recall value satisfies the user, the search is terminated at SP_A and the results are returned to the querying super-peer SP_q. If the achieved recall is not enough, then SP_A forwards the query to SP_E and, if necessary to SP_D. SP_E receives $n_{found} = 20$, $n_{other} = n_C + n_B + n_D = 35$ and computes its own results n_{local}, in order to determine the lower bound for recall as: $\frac{n_{found}+n_{local}}{(n_{found}+n_{local})+n_{other}}$. In case the achieved recall value is still insufficient, SP_q is notified about the result of the search in SP_A's subtree, and SP_q selects another neighbor (in this example SP_C) to forward

the query. The processing terminates when the desired recall value is achieved at any super-peer.

6.4 Handling Cycles

A potential problem of the proposed approach comes up when the network contains cycles. This is due to the fact that if a cycle exists, this may lead to overestimate the number of results that can be retrieved from a certain direction, due to double-counting some results. However, due to the construction phase of *SIMPEER*, each super-peer maintains for each hyper-cluster both the identifiers of the owner super-peer and the neighbor super-peer through which it received the hyper-cluster. Together with the range query $R(q, r)$, the super-peer forwards to its neighbor a list of super-peer identifiers that own the hyper-clusters. Thus, the neighbor can use this information to both avoid forwarding the query to paths that are part of a cycle, as well as avoid over-counting the number of retrievable results.

Consider again the example of Figure 7(a) and let us further assume that there exists a connection between super-peers SP_A and SP_B, thus forming a cycle. When SP_q forwards the range query to SP_A, it also sends a list of super-peers $\{SP_D, SP_E\}$, which are responsible (together with SP_A) for the estimated value of $n_A = 30$, based on the viewpoint of SP_q. Thus, SP_A takes into account only the results of SP_D and SP_E, and ignores SP_B, because indirectly SP_A knows that SP_B's results have been taken into account in n_{other} through a different path that leads from SP_q to SP_B.

7 Routing Indices Maintenance

In this section, we first discuss the maintenance cost of the routing indices. Afterwards, we describe how churn (peer joins and failures) affects the proposed framework.

7.1 Maintenance Cost

Maintenance of routing indices is required when data insertion, updates and deletions occur. Changes of the data stored at some peer do not influence the routing indices, as long as the peer's cluster descriptions do not change. In the case where a cluster description changes, then the responsible super-peer has to be informed. The super-peer updates its local index and tests if the cluster update influences the respective hyper-cluster. The other super-peers must be informed, only if a hyper-cluster of the super-peer changes. Notice that if clusters shrink, then even if the super-peers are not informed, *SIMPEER* is still able to answer correctly queries, but will probably contact more super-peers.

In practice, a super-peer informs its neighbors about changes of the hyper-clusters only if a significant change in one of its hyper-clusters is detected and it decides to broadcast this modification in a similar way to the construction phase. To summarize, data updates incur maintenance costs only if the radius of a peer cluster, and eventually its hyper-cluster, are modified.

7.2 Churn

The *SIMPEER* framework makes the system more resilient to failures compared to other P2P systems. Super-peers have stable roles, but in the extreme case that a super-peer fails, its peers can connect to another super-peer using the basic bootstrapping protocol. A peer failure may cause the responsible

super-peer to update the radii of its hyper-clusters. 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 a similarity search 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 data sets are gradually formed, with respect to the assigned super-peers.

8 Experimental Study

We evaluate the performance of *SIMPEER* using a simulator prototype implemented 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. Furthermore, we used the GT-ITM topology generator[2] to create well-connected random graphs of N_{sp} super-peers with a user-specified average connectivity (DEG_{sp}). We vary the following values: network size $N_p = 4000 - 16000$ peers, $DEG_{sp} = 4 - 7$, and $DEG_p = 20 - 60$. The number of peer clusters is $k_p = 10$, while we set the

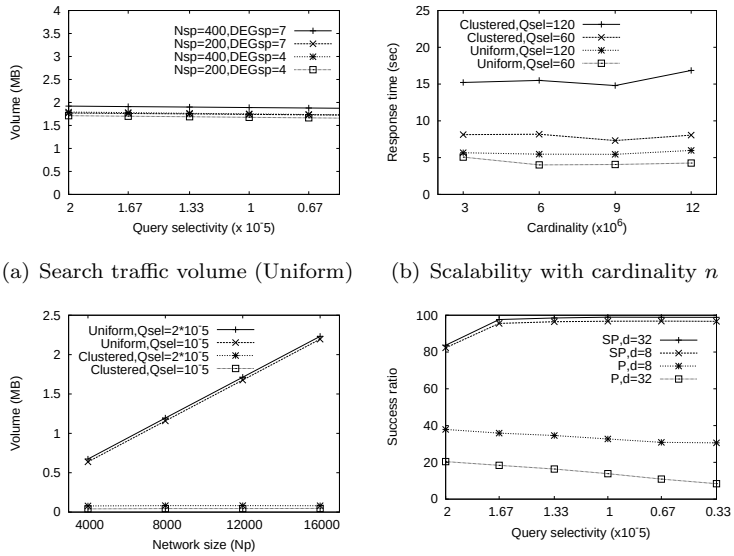
[2] Available at: http://www.cc.gatech.edu/projects/gtitm/

number of super-peer clusters as $k_{sp} = 5$. We also vary the query selectivity Q_{sel} of range queries.

In order to evaluate the scalability of SIMPEER we experimented with synthetic data collections, namely uniform and clustered, 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. 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. Again, the value of each dimension belongs to [0...10000]. We conduct experiments varying the dimensionality (3-32d) and the cardinality (1M-12M) of the dataset. 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.

8.1 Construction Cost

At first the construction cost is considered, in order to study the feasibility of the proposed routing index construction. The total construction cost is measured in terms of the total volume (in bytes) that is transferred over the network. We test two network sizes: 200 and 400 super-peers, both times with $N_p = 12000$, for varying connectivity degree ($DEG_{sp} = 4 - 7$) and for cardinality $n = 6 * 10^6$ objects. The results show that for our largest configuration, namely 400 super-peers and an average of 7 connections per super-peer, the total construction cost is approximately 600MB, which is quite tolerable.



(a) Search traffic volume (Uniform) (b) Scalability with cardinality n

(c) Scalability with network size N_p (d) Contacted SP, P with results

Fig. 8 Experiments with range queries on uniform and clustered datasets.

Practically each super-peer induces traffic equal to 1.5MB, and this cost is paid only once at construction phase.

8.2 Range Queries

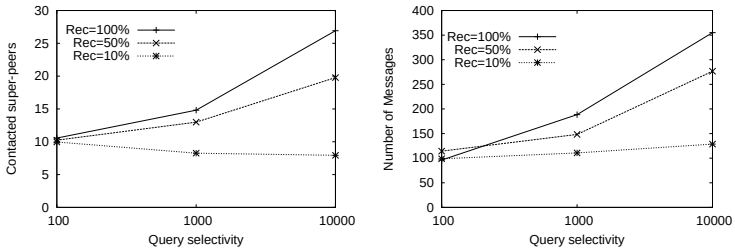
The performance of range queries is studied in the following. In Figure 8(a), we show the average traffic volume induced by range query processing for a uniform dataset. On the x-axis the query selectivity is presented as a percentage of the cardinality n. The volume decreases slightly as queries become more selective. The larger super-peer networks induce more traffic, but it is noticeable that also other topology characteristics, such as the connectivity degree of super-peers, play an important role. As the network gets denser, the cost of searching increases.

Thereafter, we studied the scalability of our approach with the cardinality n of the dataset. The results, shown in Figure 8(b), both for uniform and

clustered datasets, demonstrate that the response time increases only slightly with the dataset cardinality. Furthermore, in the case of the clustered dataset, the response time is significantly higher than for the uniform dataset. This may seem counter-intuitive, however it is due to the fact that in the uniform case many peers contribute to the results set, but only with few results. In the clustered dataset, only few peers contribute to the result set returning more objects, therefore the response time increases, since some network paths cause important delays. Figure 8(b) depicts the total response time taking into account the network delay, which depends on the size of transmitted data. We assume a modest 4KB/sec as the network transfer bandwidth on each connection.

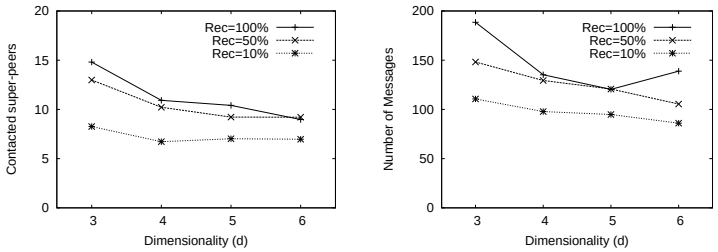
Then, we study the effect of larger networks in terms of participating peers N_p on the performance of our approach. In Figure 8(c), the traffic volume induced is depicted for uniform and clustered datasets and different query selectivity. Obviously as the network size grows, the volume in the case of uniform dataset increases too, as data objects are retrieved from any part of the network, i.e. also from farther peers. This is not the case for the clustered dataset, where the traffic volume remains practically unaffected regardless of the network size, as only specific super-peers (and network paths) contribute to the result set.

In the next experiment, we use a clustered dataset. Figure 8(d) shows the success ratio, i.e. how many of the contacted peers (P) or super-peers (SP) returned results for different dimensionality values ($d = 8$ and $d = 32$). The chart shows that specifically the super-peer success ratio is very high, reaching 98%. In the case of peer ratio, the success ratio is admittedly lower, which means that messages are sent to peers, that eventually do not contribute to the result set.



(a) Number of contacted super-peers (b) Number of messages

Fig. 9 Effect of query selectivity on range queries with recall guarantees.



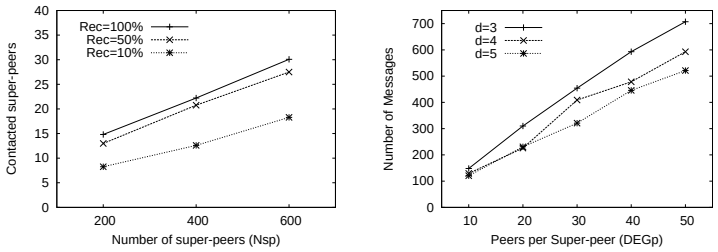
(a) Number of contacted super-peers (b) Number of messages

Fig. 10 Effect of dimensionality on range queries with recall guarantees.

8.3 Recall-based Range Queries

In this experiment, we study the performance of recall-based range queries, in terms of the number of contacted super-peers and number of messages. In the rest of the experimental study, unless explicitly mentioned, we use the default values: $N_{sp} = 200$, $N_p = 2000$, $DEG_{sp} = 4$, $d = 3$, $n/N_p = 500$ and the clustered dataset. We test different recall values, ranging from 10% to 100%.

First, we study how the range query selectivity affects the performance in Figure 9. For this purpose, we use range queries that retrieve 100, 1000 and 10000 results respectively. The chart in Figure 9(a) shows that in all cases the number of contacted super-peers is small compared to the $N_{sp} = 200$ super-peers in the network. Obviously, more selective queries require less super-peers to be contacted. Also, when the required recall is smaller than 100%, significant savings in terms of number of contacted super-peers are achieved. This verifies



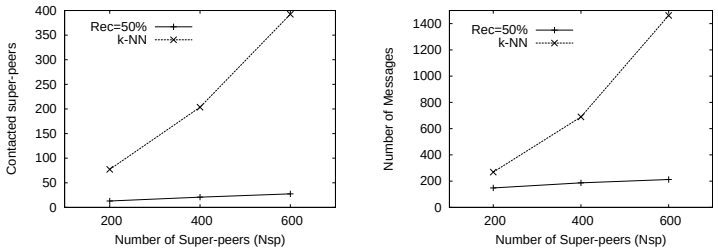
(a) Varying number of super-peers (b) Varying number of peers per super-peer

Fig. 11 Effect of network size on range queries with recall guarantees.

the validity of our framework. In Figure 9(b), we depict the associated number of messages. The chart shows that by relaxing the recall requirements, the communication cost can be drastically reduced, especially for range queries that return many objects.

Then, in Figure 10, we set the query selectivity to 1000 and we study the effect of data dimensionality. A decreasing tendency of the number of contacted super-peers is observed as dimensionality increases in Figure 10(a), due to the fact that the dataset is clustered and the data space becomes more sparse, thus less super-peers store the results for a given query. In Figure 10(b), we present the associated search cost in terms of number of messages.

In the following, we gradually increase the size of the network both in terms of N_{sp} and N_p, keeping always a constant $DEG_p = 10$. As expected, the number of contacted super-peers (shown in Figure 11(a)) increases when the size of the network increases. However, our approach scales gracefully, because we observe that when N_{sp} increases by a factor of 3 (from 200 to 600 super-peers), the number of contacted super-peers increases only by a factor of 2. This is a strong argument in favor of the scalability of our framework. In Figure 11(b), we fix the number of super-peers to 200 and we increase the number of peers per super-peer (DEG_p) from 10 to 50. Obviously, the number



(a) Number of contacted super-peers (b) Number of messages

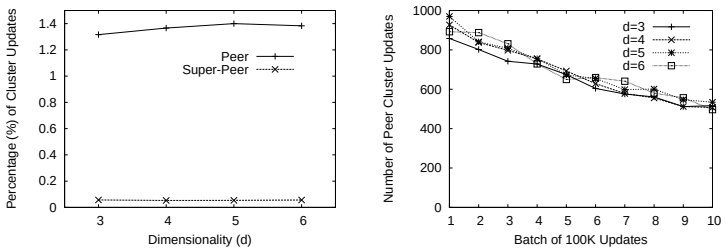
Fig. 12 Comparison of search costs for queries with recall guarantees vs. k-NN queries.

of messages required for search increases with DEG_p, as the query results are distributed over more peers.

8.4 Comparison to k-Nearest Neighbor Queries

Furthermore, we provide a discussion about recall-based range queries and their relation to k-nearest neighbor (k-NN) queries. Notice that recall-based range queries are not directly competitive to k-NN queries, since they aim at returning a small percentage of results within a range r from query point q, and not the nearest results to q. However, the strength of recall-based range queries is that their cost is significantly smaller than k-NN queries.

In Figure 12, we present a comparative study with respect to search costs. We increase the size of the network from 200 to 600 super-peers, and we keep DEG_p=10 always. We evaluate range queries with recall specified at 50% and query selectivity 1000. We also use k-NN queries with selectivity 500, which essentially return the at least as many results as the queries with recall set at 50%. Moreover, for the k-NN queries, we use the real distance of the k-th nearest neighbor, in order to avoid the extra costs related to radius estimation. The charts show that recall-based range queries are always much cheaper in terms of network costs than k-NN queries, and the benefit increases as the network size increases. This is because for k-NN queries all super-peers with



(a) (%) of cluster updates (b) Cluster updates over time

Fig. 13 Effect of cluster updates.

hyper-clusters that intersect with the k-NN query need to be contacted, while the recall-based approach contacts only a limited set of the most promising super-peers.

8.5 Data Updates

We study the effect that updates in data have on peer clusters and hyper-clusters in Figure 13. All 500 data points at each peer are updated, modeled as insertions and deletions, leading to a total of 1M data point updates. The updates follow the same data distribution as the generated data. We measure the percentage of data updates that caused an update at peer clusters and super-peer hyper-clusters for varying dimensionality (Figure 13(a)), as well as the number of peer cluster updates as time elapses (Figure 13(b)).

In Figure 13(a), the results show that less than 1.4% of data updates lead to peer cluster updates, while even less than 0.1% of data updates lead to hyper-cluster updates. Therefore only 0.1% of the updates on the peers actually cause an update at the super-peer level. In Figure 13(b), we depict how many peer clusters are updated per batch of 100K data point updates. The decreasing tendency shows that the effect of data updates diminishes as time elapses. This is due to the fact that peer clusters gradually become larger

in size, thereby accommodating a higher number of updated data, than in the beginning.

9 Conclusions

In this paper, we study efficient processing of similarity queries in metric spaces over a super-peer architecture and focus mainly on range queries. We present *SIMPEER*, a novel framework that dynamically clusters peer data, in order to build distributed routing information at super-peer level. *SIMPEER* allows the evaluation of range and nearest neighbor queries in a distributed manner that reduces communication cost, network latency, bandwidth consumption and computational overhead at each individual peer. By utilizing distributed statistics, super-peers can estimate the cardinality of the result set of a range query and guarantee a lower bound of the recall. When this recall is sufficient for the user, then the query does not have to be propagated further in the network. Finally, our experimental evaluation shows that our approach performs efficiently both in terms of computational and communication costs.

References

1. F. Banaei-Kashani and C. Shahabi. SWAM: A family of access methods for similarity-search in peer-to-peer data networks. In *Proceedings of CIKM*, pages 304–313, 2004.
2. M. Batko, C. Gennaro, and P. Zezula. A scalable nearest neighbor search in P2P systems. In *Proceedings of DBISP2P*, pages 79–92, 2004.
3. M. Batko, D. Novak, F. Falchi, and P. Zezula. On scalability of the similarity search in the world of peers. In *Proceedings of InfoScale*, page 20, 2006.
4. M. Bawa, T. Condie, and P. Ganesan. LSH forest: Self-tuning indexes for similarity search. In *Proceedings of WWW*, pages 651–660, 2005.
5. A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of SIGCOMM*, pages 353–366, 2004.

6. E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroquin. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.
7. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of VLDB*, pages 426–435, 1997.
8. P. Ciaccia, M. Patella, and P. Zezula. A cost model for similarity queries in metric spaces. In *Proceedings of PODS*, pages 59–68, 1998.
9. A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. P-tree: A P2P index for resource discovery applications. In *Proceedings of WWW*, 2004.
10. A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, and J. Shanmugasundaram. P-ring: An efficient and robust p2p range index structure. In *Proceedings of SIGMOD*, pages 223–234, 2007.
11. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of ICDCS*, pages 23– 32, 2002.
12. A. Datta, M. Hauswirth, R. John, R. Schmidt, and K. Aberer. Range queries in trie-structured overlays. In *Proceedings of P2P*, pages 57–66, 2005.
13. C. Doulkeridis, A. Vlachou, Y. Kotidis, and M. Vazirgiannis. Peer-to-peer similarity search in metric spaces. In *Proceedings of VLDB*, pages 986–997, 2007.
14. F. Falchi, C. Gennaro, and P. Zezula. A content-addressable network for similarity search in metric spaces. In *Proceedings of DBISP2P*, pages 126–137, 2005.
15. P. Ganesan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. In *Proceedings of VLDB*, pages 444–455, 2004.
16. G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, 28(4):517–580, 2003.
17. H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems*, 30(2):364–397, June 2005.
18. H. V. Jagadish, B. C. Ooi, and Q. H. Vu. BATON: A balanced tree structure for peer-to-peer networks. In *Proceedings of VLDB*, pages 661–672, 2005.
19. H. V. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang, and A. Zhou. VBI-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In *Proceedings of ICDE*, page 34, 2006.
20. P. Kalnis, W. S. Ng, B. C. Ooi, and K.-L. Tan. Answering similarity queries in peer-to-peer networks. *Inf. Syst.*, 31(1):57–72, 2006.
21. A. K.H.Tung, R. Zhangz, N. Koudas, and B. C. Ooi. Similarity search: A matching based approach. In *Proceedings of VLDB*, pages 631–642, 2006.

22. B. Liu, W.-C. Lee, and D. L. Lee. Supporting complex multi-dimensional queries in P2P systems. In *Proceedings of ICDCS*, pages 155–164, 2005.
23. D. Novak and P. Zezula. M-Chord: A scalable distributed similarity search structure. In *Proceedings of InfoScale*, page 19, 2006.
24. N. Ntarmos, T. Pitoura, and P. Triantafillou. Range query optimization leveraging peer heterogeneity. In *Proceedings of DBISP2P*, 2005.
25. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of SIGCOMM*, pages 161–172, 2001.
26. H. T. Shen, Y. Shu, and B. Yu. Efficient semantic-based content search in P2P network. *IEEE Trans. Knowl. Data Eng.*, 16(7):813–826, 2004.
27. Y. Shu, B. C. Ooi, K.-L. Tan, and A. Zhou. Supporting multi-dimensional range queries in peer-to-peer systems. In *Proceedings of P2P*, pages 173–180, 2005.
28. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM*, pages 149–160, 2001.
29. A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis. SKYPEER: Efficient subspace skyline computation over distributed data. In *Proceedings of ICDE*, pages 416–425, 2007.
30. B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of ICDE*, pages 49– 60, 2003.
31. C. Yu, B. C. Ooi, K.-L. Tan, and H. V. Jagadish. Indexing the distance: An efficient method to KNN processing. In *Proceedings of VLDB*, 2001.