Peer-to-Peer Similarity Search in Metric Spaces

Christos Doulkeridis[1]*, Akrivi Vlachou[1]*, Yannis Kotidis[1]†, Michalis Vazirgiannis[1,2]‡
[1]Dept. of Informatics, Athens University of Economics and Business, Greece
[2]GEMO Team, INRIA/FUTURS, France
{cdoulk, avlachou, kotidis, mvazirg}@aueb.gr

ABSTRACT

This paper addresses the efficient processing of similarity queries in metric spaces, where data is horizontally distributed across a P2P network. The proposed approach does not rely on arbitrary data movement, hence each peer joining the network autonomously stores its own data. 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. *SIMPEER* utilizes a set of distributed statistics and guarantees that all similar objects to the query are retrieved, without necessarily flooding the network during query processing. The statistics are employed for estimating an adequate query radius for k-nearest neighbor queries, and transform the query to a range query. Our experimental evaluation employs both real-world and synthetic data collections, and our results show that *SIMPEER* performs efficiently, even in the case of high degree of distribution.

1. INTRODUCTION

Similarity search in metric spaces has received considerable attention in the database research community [6, 14, 20]. 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. Usually objects are represented in a high dimensional feature space and a distance function, usually more complex than the L_2 norm (Euclidean distance), defines the similarity of two objects [14], such as the Levenshtein distance [21] for text retrieval or quadratic-form distances for multimedia data [28]. The computational complexity of similarity search indicates that distributed processing can share this load over a set of machines, aiming to achieve unlimited scalability, a fact also recognized in [23]. The P2P paradigm emerges as a powerful model for organizing and searching large data repositories distributed over independent sources. However, most algorithms for similarity search in metric spaces still focus on centralized settings [6, 14, 16, 33] or – lately – on structured P2P networks [23, 26] that do not necessarily preserve peer autonomy.

Super-peer infrastructures [31, 32] 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 files and queries flood the P2P network. 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 [9]) for routing queries to remote peers.

Numerous interesting applications can be deployed over such a super-peer infrastructure that supports similarity search in metric spaces. The overall objective is for a set of cooperative computers to collectively provide enhanced searching facilities, aiming to overcome the limitations of centralized settings, for example extremely high computational load. 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 relies on a super-peer infrastructure and users who wish to participate, register their machines to the P2P system. Each peer autonomously stores its own data, which is a very important feature for the aforementioned applications. Inspired by *iDistance* [16, 33] we use a one-dimensional mapping to index the data on each peer. We use the generated clusters by the iDistance method to further summarize peer data at

*Partially funded by the PENED 2003 Programme of the EU and the Greek General Secretariat for Research and Technology.
†Funded through EPEAEK II in the framework of the project 'Pythagoras II - Support of University Research Groups' with 75% from European social funds and 25% from national funds
‡Supported by the Marie Curie Intra-European Fellowship NGWeMiS: Next Generation Web Mining and Searching (MEIF-CT-2005-011549).

VLDB '07, September 23-28, 2007, Vienna, Austria.

super-peer level.

Users (at any peer) can submit queries which are propagated to an *initiator* super-peer, which then in turn routes the query selectively to those super-peers that can provide data, which belong to the query result set. Finally, the initiator gathers the objects and returns the result set to the querying peer. A major focus of this paper is how to exclude super-peers that can not contribute any results during query processing, exploiting the routing indices. We propose *SIMPEER*, a framework that supports similarity queries and provides guarantees that all similar objects to the query are retrieved, without necessarily flooding the network during query processing. We present algorithms for range and k-nearest neighbor query processing. k-NN queries are transformed into range queries, employing a radius estimation technique, using distributed statistics in the form of histograms maintained with each cluster description.

The key contributions of our work are:

- We present similarity search algorithms for metric spaces, suitable for unstructured P2P settings. Our techniques handle both range and k-NN queries.
- Our framework is based on a novel three-level clustering scheme (peer, super-peer and routing clusters) utilizing distributed statistics, in the form of histograms. These statistics are maintained by super-peers, in order to avoid processing on peers can not contribute to the result set.
- We extend a state-of-the-art centralized approach (iDistance) for similarity search in metric spaces, in order to facilitate indexing of clusters. We introduce several pruning techniques that can speed up evaluation of range queries.
- We demonstrate that, using the statistics maintained by the super-peers, k-NN queries can be transformed to simple range queries that can be, in most cases, computed efficiently in a single pass by the peers.

Section 2 provides an overview of related work and in Section 3 we describe the preliminaries. In Section 4, we present an overview of *SIMPEER*. In Section 5, we describe query processing, while routing indices are described in Section 6. Thereafter, k-NN search is examined in detail in Section 7. The experimental results are presented in Section 8, and finally we conclude in Section 9.

2. RELATED WORK

Similarity search in P2P systems has attracted a lot of attention recently, however most approaches focus mainly on structured P2P systems or on building a network topology that groups together peers with similar content.

Recently, MCAN [11] and M-Chord [23] were proposed to handle the metric similarity search problem in structured P2P networks. 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 [16] 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 [30] respectively. It is worth noticing that data preprocessing (clustering and mapping) is done in a centralized fashion, and only then data is assigned to peers. This constitutes an important difference to our framework.

Recent work aims to process similarity search in P2P systems by building a suitable network topology. A general solution for P2P similarity search for vector data is proposed in [2], named SWAM. Unlike structured P2P systems, peers autonomously store their data, and efficient searching is based on building an overlay topology that brings nodes with similar content together. However, SWAM is not designed for metric spaces. A P2P framework for multidimensional indexing based on a tree structured overlay is proposed in [18]. LSH forest [3] stores documents in the overlay network using a locality-sensitive hashing function to index high-dimensional data for answering approximate similarity queries.

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 [29]. The use of Skip Graphs for range query processing has also been proposed in [12]. Several P2P range index structures have been proposed, such as Mercury [4], P-tree [8], BATON [17]. 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. Recently, in [19], routing indices stored at each peer are used for P2P similarity search. Their approach relies on a freezing technique, i.e. some queries are paused and can be answered by streaming results of other queries.

While there exists some work on P2P similarity search that focuses on caching [13, 27] or replication [5], our work's primary focus is query processing. Obviously, *SIMPEER* can be enhanced with caching mechanisms such as the above or by exploiting overlay topologies that cluster peers with similar contents [10].

3. PRELIMINARIES

In this section we present a brief introduction of similarity searching in metric spaces and we describe the query types that should be supported. Further, an efficient approach for centralized systems, namely iDistance [16, 33] is presented, since *SIMPEER* extends its basic concepts.

3.1 Metric Space and Query Types

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).

In this paper instead of referring to the feature values of an object, we refer to the object itself. There are two types of similarity queries:

range query $R(q,r)$ Retrieve all elements that are within

distance r to q. This is $\{u \in U : dist(q,u) \leq r\}$.

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

3.2 iDistance

iDistance [16, 33] is an index method for similarity search. It partitions the data space into n clusters and selects a reference point K_i for each cluster C_i. 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 in cluster i are mapped to the interval $[i * c, (i+1) * c]$, as shown in Fig. 1.

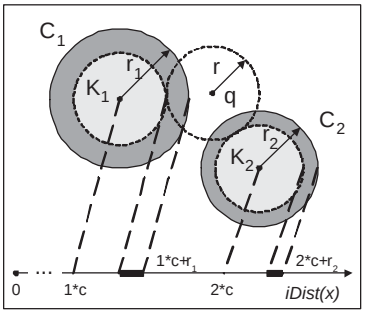


Figure 1: *iDistance* mapping to 1-dimensional values.

The actual data objects are stored in a B^+-tree using the iDistance values as keys. Additionally, the cluster centers K_i and the cluster radius r_i are kept in a main memory list. In this way, the problem of similarity search is transformed to an interval search problem. For a range query $R(q,r)$, for each cluster C_i that satisfies the inequality $dist(K_i,q) - r \leq r_i$[1], the data elements that are assigned to the cluster C_i and their iDistance values belonging to the interval $[i * c + dist(K_i,q) - r, i * c + dist(K_i,q) + r]$ are retrieved. For these points p_i the actual distance to the query point is evaluated and thereafter, if the inequality $dist(p_i,q) \leq r$ holds, p_i is added to the result set. The algorithm proposed in [33] for nearest neighbor search is based on repetitive range queries with growing radius.

4. SYSTEM OVERVIEW

The overall aim is to provide an infrastructure for answering similarity queries in metric spaces in super-peer networks. More formally, we assume a P2P network 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 network, i.e. $N_{sp} << N_p$. Peers that join the network directly connect to one of the super-peers. Each

[1]Henceforth mentioned as intersection of the range query $R(q,r)$ and the cluster C_i.

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
c	iDistance constant
$dist()$	Distance function
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
$R(q,r)$	Range query
$NN_k(q)$	k-nearest neighbor query

Table 1: Overview of symbols.

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 ($DEG_{sp} < DEG_p$). An overview of the symbols used can be found in Table 1. In this paper, we propose an approach that enables similarity search in metric spaces over data distributed in a super-peer network, utilizing routing indices based on cluster information.

In *SIMPEER* 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. In order to make its data searchable by other peers, each peer first clusters its data using a standard clustering algorithm (like K-Means), and then sends the cluster descriptions C_i, namely the cluster centroid and radius (K_i,r_i), to its super-peer. Only the cluster descriptions C_i as a summarization of the peers' data is published to the super-peer, while the original data is stored by the peer. The iDistance method is employed by the peer to index and provide access to its data, in order to efficiently answer similarity queries during local query processing.

Each super-peer SP_A maintains the cluster descriptions of its associated peers. 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 *hyper-clusters*, which summarize the data objects of all peers connected to the super-peer. The super-peer keeps a list of the hyper-clusters in main memory and stores in a B^+-tree the peers' clusters using an extension of the iDistance technique that is capable to handle cluster descriptions instead of data points. This extension, introduced in the next section, enables efficient similarity searching over the cluster descriptions, so that the query is posed only to peers having data that may appear in the result set.

The remaining challenge is to answer such 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 and it is described in detail in Section 6. The number of

collected hyper-clusters can be potentially large, therefore the super-peer applies a clustering algorithm that results in a set of *routing clusters*, that constitute a summary of the hyper-cluster information. In a completely analogous way to the indexing technique of the peers' clusters, the super-peer uses the proposed extension of iDistance to store the hyper-cluster information, this time maintaining in main memory only the routing clusters.

To summarize, *SIMPEER* utilizes a three-level clustering scheme:

- Each peer clusters its own data. 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.
- 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.

Our routing indices based on cluster summarization support efficient query processing, in terms of local computation costs, communication costs and overall response time, of both range and k-NN queries. More detailed, a query may be posed at any peer and is propagated to the associated super-peer, which becomes responsible for the query processing and finally returns the result set to the querying peer.

Given a range query $R(q, r)$, each super-peer SP_A that receives the query uses its routing clusters to forward the query to the neighboring super-peers which have either locally or in their routing indices clusters that intersect with the query (Section 6.1). Thereafter, SP_A forwards the range query only to its own peers that have clusters intersecting with the query based on the hyper-clusters (Section 5), 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 (or peer in the case of the initiator) from which SP_A received the query.

To process nearest neighbor queries, the initiator super-peer is responsible to map this query to a range query and then propagate it to the neighboring super-peers based on its routing index. One of the arising challenges is the transformation of a nearest neighbor query to a range query. A k-NN query $NN_k(q)$ is equivalent to a range query $R(q, r_k)$ where r_k is the distance of the k-th nearest neighbor from the point q. The main problem is that the distance r_k is not known a priori. Therefore, a heuristic is required to estimate the distance of the k-th nearest neighbor. In this paper we propose two alternatives (Section 7) to estimate an appropriate range over the distributed data, based on histograms that capture distance distributions within clusters.

5. DISTRIBUTED QUERY PROCESSING

In this section, we focus on the query processing performed by each super-peer based on its peers' data. In more detail, first we consider the query processing performed by a single peer, therefore we present the iDistance indexing technique used by each peer. Then, we consider query processing with respect to a single super-peer and its associated peers. For this task, an efficient extension of iDistance is proposed and a search algorithm is presented that allows a super-peer to choose the subset of its peers that are relevant to the query. Finally, we discuss the extensions that are required to support k-NN queries and focus on the progressive evaluation of range queries with increasing radius. In the next section, we present the routing indices technique, that enables efficient similarity search over the whole network.

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
```

5.1 Peer Query Processing

Each peer is responsible for its own data, which is organized and stored based on the iDistance concept. 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 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 and a radius r_i, which is the distance of the farthest point of the cluster to the centroid.

Each data object is assigned to the nearest cluster C_i and it is mapped to a one dimensional value following the same mapping as iDistance. The iDistance values of the data objects are indexed in an ordinary B^+-tree, while the list of the clusters LC_p, with the centroids and the radii of the clusters, is kept in main memory.

Peers process mainly range queries over their local data. As stated in [16, 33], the range algorithm 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 cluster area that should be scanned in order 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 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 Fig. 1 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.

Notice that in contrast to [33] we do not focus on I/O optimization issues, since it is out of the scope of this paper. Additionally, B^+-trees are available in any commercial database system, so that our algorithm can be built on any existing system and peers do not have to maintain special purpose indexes.

5.2 Super-peer Query Processing

A super-peer SP_A processes a range query by using its peers' cluster descriptions. Therefore, a super-peer determines 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. Since the number of clusters increases rapidly according to the number of connected peers, in order to reduce the number of distance computations and intersection calculations and provide efficient query processing, the iDistance concept is followed. SP_A applies a clustering algorithm on the cluster descriptions and – in a similar way to iDistance – maps high-dimensional points to one dimensional values. The cluster descriptions (having high-dimensional cluster centers) are mapped to one dimensional values, in such a way that range and k-NN queries can be mapped into an interval search.

In the following, first we extend the iDistance mapping for clusters and then we present the search algorithm, which ensures that all clusters that intersect with the range query are retrieved.

5.2.1 One Dimensional Mapping of Clusters

A super-peer SP_A collects the cluster descriptions from its associated peers LC_{sp}=$\{(K_1, r_1), ..., (K_{n_{sp}}, r_{n_{sp}})\}$, where n_{sp} is the total number of clusters. 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 clusters (called 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 center and r'_i the hyper-cluster radius, which is the distance of the farthest point of all clusters assigned to the hyper-cluster, to the centroid.

Each cluster C_j is mapped to a one-dimensional value based on the nearest hyper-cluster center 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. In more detail, 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 distances practically define the effective data region of reference point O_i, or in other words, the region where all points of all clusters C_j belong to.

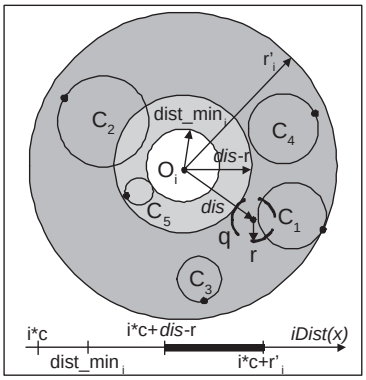


Figure 2: Covering region for a hyper-cluster O_i, and search interval based on range query $R(q, r)$.

Algorithm 2 Range Query Search.

```
1: Input: (q, r)
2: Output: Result set S
3: for O_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:       lnode ← LocateLeaf(btree, i * c + lower)
8:       Outward(lnode, i * c + r'_i, dis)
9:    end if
10: end for
11: return S
```

5.2.2 Range Query Processing

In this section, we provide an algorithm that retrieves all clusters that intersect with a given range query $R(q, r)$. 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 points 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 Fig. 2 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. 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 (Fig. 2), 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 $lower = max(dis-$

Algorithm 3 Outward.

```
1: Input: (node, ivalue, dis)
   /* {E} is the set of entries in node */
2: for (e_i : (key_i, K_i, r_i, d_i, IP_i) ∈ {E}; key_i < ivalue) do
3:   if (|dis − d_i| ≤ r + r_i) then
4:     if (dist(K_i, q) ≤ r + r_i) then
5:       S ← S ∪ e_i
6:     end if
7:   end if
8: end for
9: if (e_last.key < ivalue) then
10:   Outward(node.rightnode, ivalue)
11: end if
12: return
```

$r, dist\_min_i$) (line 5), for starting an interval search on the B^+-tree (line 7), and 3) scans the interval until r'_i (line 8). *LocateLeaf()* implements a standard search for some input value on a B^+-tree and returns the leaf node corresponding to the input value. Notice that we use the same notation as in the original iDistance publication [33].

Outward (Algorithm 3) takes as input the leaf node from which the search starts, the high-end value for searching and the distance *dis* of the query point to the hyper-cluster. Each entry e_i, i.e. cluster, stored in node is checked to ensure that it is within distance $r+r_i$ to the query (line 4). The objective of the range query algorithm is to reduce, besides the number of accessed nodes, also the number of distance computations needed for query processing, which may be costly for metric spaces and complex distance functions. For this propose, all the information concerning distances stored in the nodes of the B^+-tree is used to effectively apply the triangle inequality. The $|dis - d_i|$ (line 3) is a lower bound on the distance of $dist(K_i, q)$. The inequality in line 3 reduces the number of distance computations during query processing. If necessary, the algorithm initiates a search to the next leaf (line 9-10) of the B^+-tree.

5.3 k-NN Search

In a similar way to iDistance, peers are capable to process nearest neighbor queries locally. According to [16, 33], a peer would process the query by executing a range query with an initial range. If less than k data objects are retrieved, the radius of the range query is increased repeatedly, until k data object are retrieved. Since our application area is a P2P environment, a strategy that uses a small radius and increments it until k objects are retrieved would cause more than one round-trips over the network, which is quite costly. In *SIMPEER* the super-peers maintain a set of statistics that can estimate the distance of the k-th object from the query point and avoid the execution of multiple range queries. In Section 7, we discuss alternative radius estimation techniques and also a naive evaluation over the initiator's peers, to estimate an upper bound of the required range.

In our approach the evaluation of nearest neighbor queries is restricted to at most two round trips. In the first round trip, a range query is posed with the radius estimated by the distributed statistics. If the first round trip fails to retrieve k objects, a second round-trip cannot be avoided. In the second round trip, the upper bound defined by the naive evaluation over the initiator's peers is used, in order to ensure that at least k objects are retrieved. To reduce the costs of the second round trip (if required), our range query algorithms are extended to efficiently support the progressive evaluation of range queries with increasing radius. The range query $R(q, r)$ is enhanced with a lower bound of the distance r_{low}, i.e. $R(q, r, r_{low})$ and the proposed algorithms support the evaluation of such queries by retrieving all objects p having a distance $r_{low} \leq dist(p, q) \leq r$. The condition of Algorithm 2 (line 6) is enhanced with the condition $dis + r'_i \geq r_{low}$ in order to filter out hyper-clusters that cannot contribute to the result set. Similarly, in Algorithm 3 (line 3) the condition $dist(e_i, q) + r(e_i) \geq r_{low}$ is included to discard clusters that were retrieved in the previous round trip. Finally, Algorithm 1 is modified by posing two interval searches on the B^+-tree, namely $[dist(K_i, q)+i*c-r, dist(K_i, q)+i*c-r_{low}]$ and $[dist(K_i, q) + i * c + r_{low}, dist(K_i, q) + i * c + r]$.

In order to further reduce the network traffic for the case of k-nearest neighbor search, each intermediate super-peer receives at most k results from each neighboring super-peer or connected peer and propagates back (using the query routing path) only the k results with lowest distance to the query. It should be noted, that to reduce the associated computation and routing costs even more, intermediate super-peers can decrease the estimated radius, provided that they find k results locally and the distance of the k-th object is smaller than the estimated radius. Nevertheless, this would require serialized query processing, therefore we do not consider this option.

6. ROUTING INDICES

This section first discusses the query routing that is performed by utilizing the routing indices and presents the construction and maintenance of the routing information at super-peer level. Afterwards, we discuss how churn (peer joins and failures) affects the proposed system.

6.1 Usage, Construction and Maintenance

As regards routing index 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. In more detail, 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. 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:

$$dist(O_i, q) > r + r_i$$

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. Even though this broadcasting phase can be costly, especially for very large super-peer networks, we emphasize that this cost 1) is paid only once at construction time, 2) depends mainly on the number of super-peers N_{sp} and hyper-clusters per super-peer k_{sp} and not on the cardinality n of the data set, as in the case of structured P2P networks, and 3) can be tolerated for the network sizes of currently deployed super-peer

networks, as we will also show in the experiments. In this paper we focus on algorithms that return accurate results in the covered network. We leave for future work the extension and study of query processing in the cases of networks with limited resources that can not support broadcasting of super-peer hyper-clusters.

Because the list of assembled hyper-clusters at SP_r may be potentially big to maintain in main memory, SP_r runs a clustering algorithm, which results in a set of *routing clusters (RC)*. 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 the extension of the 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 of iDistance, analogously to the technique employed for its peers' clusters.

Maintenance of routing indices is straightforward and can be accomplished using the well-established techniques described in [9]. In practice, when a super-peer detects a significant change in one of its hyper-clusters, it decides to broadcast this modification in a similar way to the construction phase. A modification can be due to peer data updates that altered the radius of a peer cluster, and eventually its hyper-cluster. It can also be the result of churn (peer joins or failures), that alter the radius of a hyper-cluster.

6.2 Churn

The *SIMPEER* architecture 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 its hyper-cluster radius. Only if churn rate is high, these changes are propagated to other super-peers. Even if updates do not occur 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. This is close to the notion of semantic overlay network (SON) construction [10], which is similar to having clustered data assignment to super-peers.

7. NEAREST NEIGHBOR SEARCH

In highly distributed systems, such as P2P networks, each communication phase (round trip) during query processing is costly. Ideally, in a P2P environment, it is beneficial to have a small fixed number of round-trips in the network, ensuring small query response times, even at the cost of higher local computation costs and local interactions.

In this section, we present our approach in which the super-peers maintain a set of statistics that aim to estimate the distance of the k-th object from the query point and avoid the execution of multiple range queries. Given a k-NN query q, the objective is to find a radius r_k to transform $NN_k(q)$ into a range query $R(q, r_k)$. To avoid multiple round trips, the radius r_k should be slightly overestimated, in order to increase the probability that one round-trip is enough for retrieving the correct results. If less than k data objects are retrieved, a second round trip can not be avoided. To ensure that in the second round trip we return at least k objects, we present an algorithm for the naive evaluation of k-NN queries, with respect only to the initiator super-peer and its associated peers, in order to estimate an upper bound of the required range for a distributed k-NN query.

To summarize we consider two different methods for radius estimation that can be performed at any initiator super-peer locally, i.e. without any communication, namely: 1) *histogram-based local estimation (LE)*, and 2) *histogram-based global estimation (GE)*. A histogram maintains information about the distribution of distances within a cluster. We also present a naive approach, called *initiator computation (IC)*, in which the initiator actually communicates with all its peers to retrieve a local radius value, which is used as an upper bound for radius.

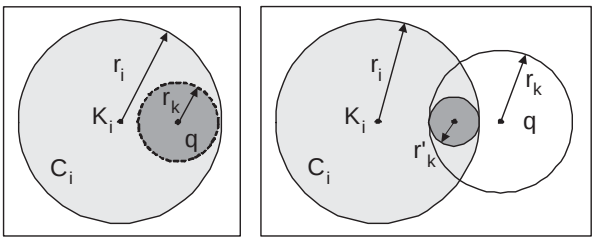
7.1 Initiator Computation

The initiator super-peer SP_A determines the nearest cluster $C_i \in LHC_A$ to the query point q, where LHC_A denotes the main memory list of hyper-clusters. Thereafter, SP_A determines the peer to which the cluster C_i belongs, and sends a k-NN query to it. As mentioned before, peers are capable to process k-NN queries, for instance by repeatedly executing local range queries with increasing range[2]. After processing the query, the peer returns the distance r_k of its k-th nearest object to the query. The distance r_k can be used by SP_A to initiate the range query $R(q, r_k)$ that corresponds to the k-NN query. Notice that in this approach, the distance r_k is large enough to ensure us that at least k objects are retrieved and no other round-trip is required. In fact, r_k can be potentially large, since it is computed based on the data present on one peer only. Therefore SP_A needs to shrink the range of the posed query, however without paying a high cost (e.g. by contacting other super-peers). An option is to send a range query $R(q, r_k)$ to the rest of its peers only, in order to retrieve a better (i.e., smaller) value of r_k. This value is used to query the rest of the super-peer network. It should be emphasized that initiator computation requires actual communication between the initiator and its peers, so it is a distributed process that bears some non negligible communication cost, in contrast to the proposed estimation techniques.

7.2 Histogram Construction

Our approach is applicable for any metric-based distance function, thus, it does not rely on information about data

[2]If more advanced local algorithms are employed this is complementary to our work



(a) $dist(K_i, q)+r_k \leq r_i$ (b) $dist(K_i, q) + r_k > r_i$

Figure 3: Radius estimation r_k for query (q, r_k).

distribution. Metric spaces are characterized by (an estimate of) the distance distribution of the objects [7].

Given a metric space M the (overall) distribution of distances is defined as: $F(r) = Pr\{d(q,p) \leq r\}$, where q and p are any two random points. Since $F(r)$ denotes the probability for a point to lie in the range query with radius r and center the point q the expected number of objects retrieved by a range query $R(q,r)$ are $objs(R(q,r)) = n * F(r)$, where n denotes the cardinality of the dataset.

We follow the same methodology as in [7]. We assume that the index of homogeneity of viewpoints is "high", which means that two random points are very likely to have an almost common view of the metric space M. [7] verifies that this is very common in a wide variety of real and synthetic datasets. Capitalizing on this assumption, the relative distance distribution F_q of a query object q is well approximated by a sampled distance distribution $\hat{F}$, which can be represented by the $m \times m$ matrix of pairwise distances between m objects.

We propose an approach relying on histograms to approximate the distance distribution. Histograms are employed to summarize the number of distances between data object. Optimization of histograms is out of the scope of this paper and is left as future work, so we base our model on equi-width histograms, even though we could succeed better performance with optimized histograms [1, 15].

Our objective is to represent the approximate distance distribution within each cluster C_i and estimate the distance of the k-th nearest object to a query point q. Towards this goal, the distance distribution is approximated by an equi-width histogram with size of bin r_B[3], respectively storing the values of: $\hat{F}_i(r_B)$, $\hat{F}_i(2 * r_B)$ ... $\hat{F}_i(s * r_B)$, i.e., using as many bins (s) as necessary for reaching the maximum distance in cluster C_i, in other words the minimum s for which: $s * r_B \geq 2 * r_i$. In fact, each bin $\hat{F}_i(l * r_B)$ holds the frequency of the distances for which: $dist(p_m, p_n) \leq l * r_B$ where p_m, p_n any data objects.

7.3 Histogram-based Radius Estimation

We first propose a method (LE) for estimating the distance of the k-th nearest neighbor based only on the cluster descriptions kept by one super-peer. For any cluster C_i that intersects with the query $R(q, r_k)$, we estimate the number of objects that can be retrieved by the query based on C_i's histogram. Assuming a radius r_k, for any intersecting cluster C_i two cases can be distinguished: 1) the inequality

[3]In the following, for sake of simplicity we assume that cluster histograms are represented with a common size of bins r_B.

Algorithm 4 k-NN radius estimation.

```
1: Input: Query q, k, List of peer clusters LC_p, Maximum
   histogram bin of any cluster C_i: s_max
2: Output: Estimated radius r_k
3: low ← 0
4: high ← s_max
5: while (low < high) do
6:    mid ← ⌈(low+high)/2⌉
7:    sum ← 0
8:    for (C_i ∈ LC_sp) do
9:       if (dist(K_i, q) + mid ≤ r_i) then
10:         sum ← sum + n_i * F̂_i^n(mid * r_B)
11:      end if
12:      if (dist(K_i, q) + mid > r_i) then
13:         sum ← sum + n_i * F̂_i^n((mid*r_B+r_i−dist(K_i,q))/2)
14:      end if
15:   end for
16:   if (sum ≥ k) then
17:      high ← mid
18:   else
19:      low ← mid
20:   end if
21: end while
22: r_k ← low * r_B
23: return r_k
```

$dist(K_i, q) + r_k \leq r_i$ holds, or 2) cluster C_i partially intersects with the range query $R(q, r_k)$. An example of both cases for the Euclidean distance is depicted in Fig. 3.

For the first case we estimate the number of objects (o_i) retrieved by the query $R(q, r_k)$ from cluster C_i:

$$o_i = objs(R(q, r_k)) = n_i * \hat{F}_i(r_k) \quad (1)$$

where n_i is the cardinality of cluster C_i.

In the second case, where the query partially intersects with a cluster, the number of objects (o_i) retrieved by the query $R(q, r_k)$ from cluster C_i is estimated based on a smaller radius that corresponds to the intersection and is approximated by $r'_k = \frac{r_i + r_k - dist(K_i,q)}{2}$. The estimated number of objects (o_i) retrieved from C_i is:

$$o_i = objs(R(q, \frac{r_i + r_k - dist(K_i, q)}{2})) \quad (2)$$

Considering Fig. 3(b), we use as query area the dark grey shadowed circle that is contained in the cluster, which is smaller than the real intersection, and leads to an underestimation of the number of objects contained, therefore an overestimation of radius r_k.

Algorithm 4 describes how radius estimation is performed. The intuition of the algorithm is based on a binary search of the interval $(low, high) = (1, s_{max})$. Notice that s_{max} denotes the maximum number of bins in any histogram of all clusters C_i. In more detail, in each iteration of the algorithm (line 5), the middle mid of the interval $(low, high)$ is used, in order to search within each intersecting cluster C_i. If C_i contains the query area, equation 1 is used, otherwise equation 2 is employed, in order to add up the estimated number of objects that can be retrieved from intersecting clusters. If the estimated number of objects (sum) is larger than or equal to k (line 16), a binary search on the interval (low, mid) is initiated. If the number of objects is less than k (line 18) the interval searched is $(mid, high)$. Eventually,

the binary search terminates, based on the condition on line 5, and the estimated value of the radius (line 22) is returned.

In Algorithm 4, we implicitly assume that all clusters in LC_{sp} intersect with the query. In order to reduce the computational cost, instead of examining all clusters $C_i \in LC_{sp}$, we first examine whether the hyper-cluster $HC_i \in LHC$ intersects with the query. For each hyper-clusters HC_i which intersects with the query, we retrieve from the B^+-tree the clusters intersecting with the query, based on their iDistance values.

7.4 Global Radius Estimation

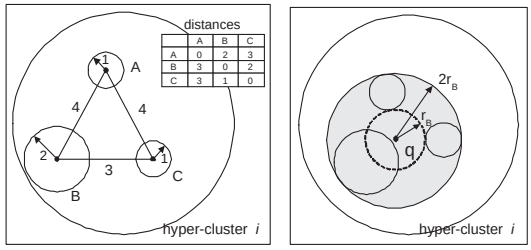
In order to get a more refined radius estimation, we extend the estimation algorithm to be applicable over the routing indices that describe the objects over the whole network. Thus, we consider how to enrich the hyper-clusters, which are stored for routing purposes, with histogram information aggregated from the individual clusters.

Let us consider a hyper-cluster HC_i where a set of clusters C_j – each of them having a histogram describing its distance distribution – are assigned to HC_i. Each hyper-cluster is enhanced with two histograms, in order to estimate the number of data objects retrieved by a range query from HC_i. In other words, the distance distribution is described by a two-level histogram approximation. The first histogram hc_i determines the number of clusters nc_i that the query intersects. The second histogram hd_i maintains an estimate of the number of data objects nd_i that a cluster contains, based on the radius of the intersection.

Practically, for histogram hc_i, we need to describe the distance distribution of the clusters within the hyper-cluster. We build a histogram based on the cluster distances. Similarly to the histogram construction of the cluster, we approximate the distance distribution based on the frequencies of distances that occur for any pair of clusters. Let's assume that we have two clusters C_i and C_j. We use as distance between two cluster centroids K_i, K_j the distance of the cluster centroid K_i to the nearest point of cluster C_j, i.e. $dist(K_i, K_j) - r_j$. In other words, we calculate the required radius of a range query at $q = K_i$ for intersecting with cluster C_j. For example consider Fig. 4(a) that depicts three clusters A, B and C, inside a hyper-cluster and a matrix with their pairwise distances $dist(K_i, K_j) - r_j$. The matrix is read row-by-row, for instance the first row means that cluster's A distance to B is 2 units, while its distance to C is 3 units. If r_B denotes the histogram's bin size, the value of bin i is then computed as the number of cells in matrix with value smaller or equal to i divided by the total number of cells. In this example, the histogram values are $r_B = 4/9$, $2 * r_B = 6/9$ and $3 * r_B = 1$.

Then, for histogram hd_i, we need to summarize the information of clusters C_j. Since our goal is to estimate a radius that probably contains k points in its range, HC_i keeps a new superimposed histogram hd_i containing in each bin the minimum value of the bins of all cluster histograms that correspond to the same distance. In addition, for each hyper-cluster HC_i we also maintain the minimum cardinality $n_c = min(n_{C_j} | C_j \in HC_i)$ of all clusters belonging to HC_i. The summarized histogram and the minimum cluster cardinality n_c are attached to the hyper-cluster HC_i, and broadcasted during the routing index construction phase.

Now we can proceed to global radius estimation, based on the two histograms. The algorithm for the estimation



(a) distance calculation (b) radius estimation

Figure 4: Hyper-clusters for radius estimation.

based on the hyper-clusters through the routing index is completely similar to the Algorithm 4. Again, our algorithm is based on a binary search to estimate the radius r_k. In each step, a potential radius r_k is calculated and all hyper-clusters $\{HC_i : (O_i, r'_i)\} \in LHC$ that intersect with the query based on the current radius r_k are retrieved. For each hyper-cluster the number of objects o_i retrieved by the query $R(q, r_k)$ from hyper-cluster HC_i is estimated. In a similar way to the local estimation, we distinguish two cases and define as l the number of bins that are required for intersection. If the inequality $dist(O_i, q) + r_k \leq r'_i$ holds we define $l = \lceil \frac{r}{r_B} \rceil$ else $l = \lceil \frac{r'_i + r_k - dist(O_i,q)}{2 * r_B} \rceil$. The number of objects o_i retrieved by a range query $R(q, r_k)$ from the hyper-cluster HC_i is given by: $o_i = \sum_{j=0}^{l} (nc_i(j * r_B) * nd_i(j * r_B))$, where $nc_i(j * r_B)$ is the estimated number of clusters that intersect the query with radius $j * r_B$ (and do not intersect with radius $(j-1) * r_B$) and $nd_i(j * r_B)$ is the estimated number of objects retrieved by a query with radius $j * r_B$. The number of clusters nc_i that intersect with the query is given by: $nc_i(j) = n_{HC_i} * (hc_i(j * r_B) - hc_i((j-1) * r_B))$, while the number of objects is given by $nd_i(j) = n_c * hd_i((l-j) * r_B)$, where $j \leq l$ and $hc_i(-1) = 0$. For the example in Figure 4(b) the number of points for $r_k = 2 * r_B$ results in: $2 * hd_i(r_B) + hd_i(2 * r_B)$. Finally, we are looking for the smallest radius r_k for which the inequality $\sum_{HC_i \in LHC} o_i \geq k$ holds.

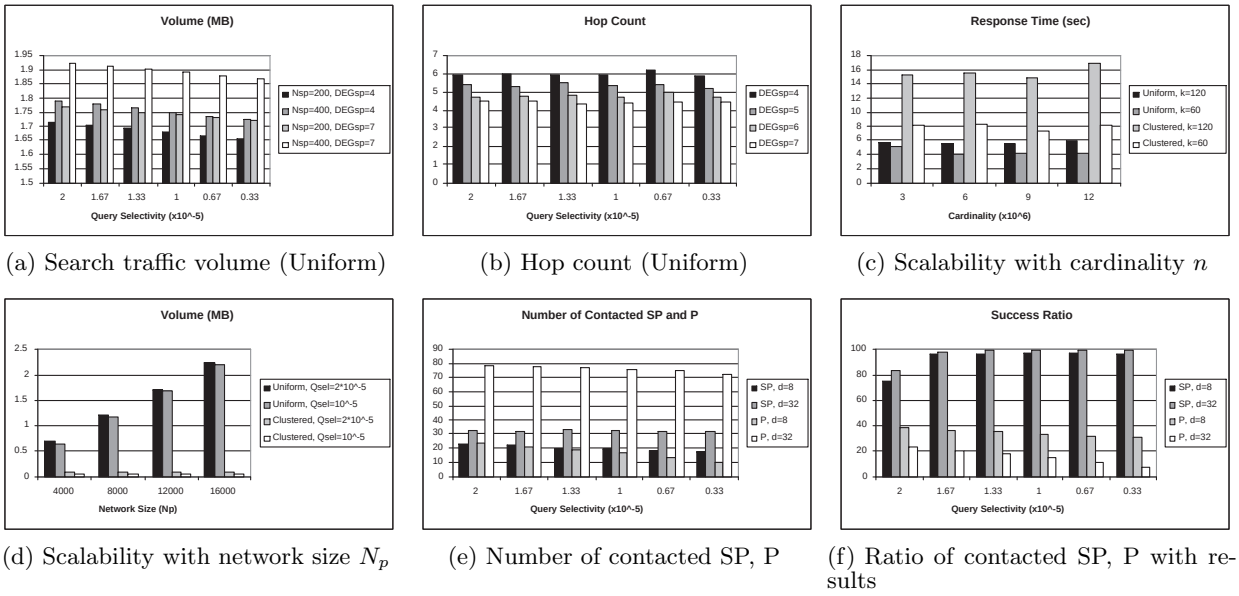
8. EXPERIMENTAL STUDY

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.

In our experiments, we used the GT-ITM topology generator[4] 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 test the effect of different super-peer clusters $k_{sp} = \{5, 10, 15\}$. We also change 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

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



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

(d) Scalability with network size N_p (e) Number of contacted SP, P (f) Ratio of contacted SP, P with results

Figure 5: Experiments with range queries on uniform and clustered datasets.

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 (8-32d) and the cardinality (3M-12M) of the dataset. Additionally, two real data collections were used: 1) VEC: consists of 1M 45-dimensional vectors of color image features, and 2) Covtype consists of 581K 54-dimensional instances of forest Covertype data, available from UCI Machine Learning Repository.[5] 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. By means of a simple analysis, it is straightforward to derive that the construction cost depends mainly on the super-peer topology, in other words on the number of super-peers (N_{sp}) and the number of super-peer clusters (k_{sp}).

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

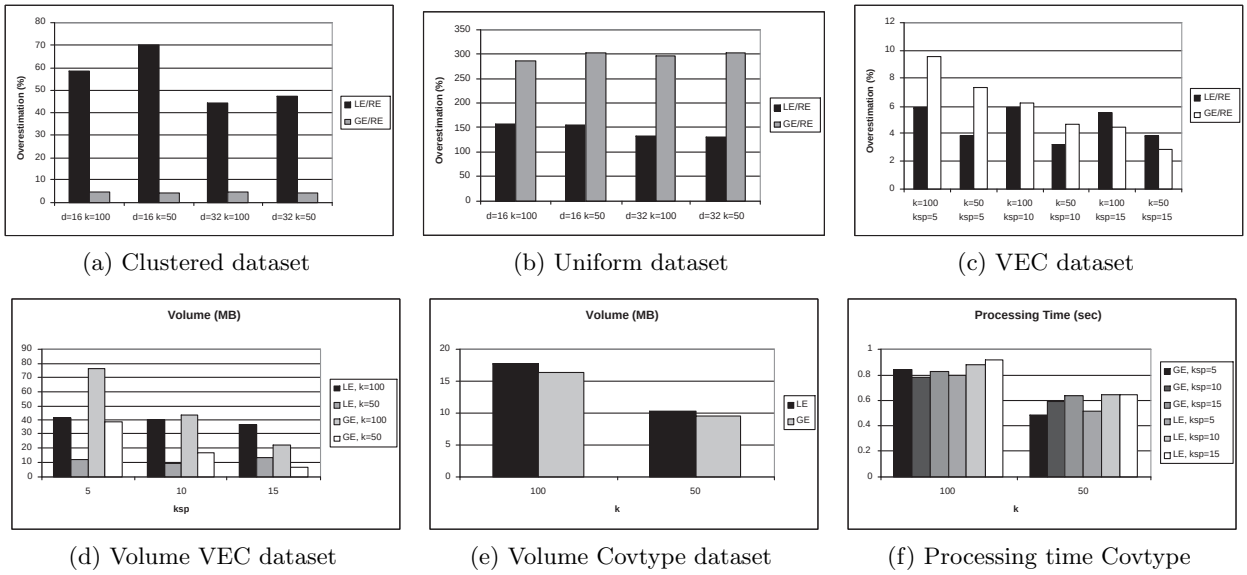
[5]http://www.ics.uci.edu/ mlearn/MLRepository.html

8.2 Range Queries

The performance of range queries is studied in the following. In Fig. 5(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. Clearly the volume decreases 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. The maximum hop count decreases for higher DEG_{sp} values, as depicted in Fig. 5(b). This is the number of maximum number of hops required to reach the most distant peer that contributes results to the final result set.

Thereafter, we studied the scalability of our approach with the cardinality n of the dataset. The results, shown in Fig. 5(c), both for uniform and clustered datasets, demonstrate that the response time increases only slightly with the dataset cardinality. In Fig. 5(c), we denote as k the selectivity of the range query in terms of retrieved objects. 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. Fig. 5(c) 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 Fig. 5(d), the traffic volume induced is depicted for uni-



(a) Clustered dataset (b) Uniform dataset (c) VEC dataset

(d) Volume VEC dataset (e) Volume Covtype dataset (f) Processing time Covtype

Figure 6: Comparison of radius estimation techniques and results from k-NN search.

form 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 set of experiments, we use a clustered dataset. Fig. 5(e) shows the number of contacted super-peers (SP) and peers (P) for different dimensionality values ($d = 8$ and $d = 32$). Fig. 5(f) shows the success ratio, i.e. how many of the contacted peers (super-peers) returned results. 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.

8.3 k-Nearest Neighbor Queries

For nearest neighbor queries, we first evaluate the radius estimation techniques, and then we proceed to study the query processing cost. Fig. 6(a), 6(b), 6(c) show our experimental results for radius estimation. The setup for synthetic datasets is $N_{sp} = 200$, $N_p = 4000$ and $n = 10^6$, while for the real datasets we use $N_p = 2000$. Given a nearest neighbor query $NN_k(q)$, we use RE to refer to the real distance of the k-th nearest neighbor, while LE refers to local estimation and GE to global estimation. For combination X/Y the overestimation percentage is computed as: $100 * \frac{X-Y}{X}\%$.

Our results show that in the case of clustered datasets GE performs better (Fig. 6(a)) than LE. However for uniform datasets, LE is better than GE (Fig. 6(b)), as all peers have objects that are nearby to any query point. Thus LE works well for any super-peer. In the case where the initiator super-peer does not index objects near the query point, LE fails and overestimates highly the radius. On the other hand, GE is influenced by the clustering quality and the number of the hyper-clusters k_{sp}. Therefore the performance of LE is more stable for uniform data, than the GE technique.

In the case of the VEC dataset (Fig. 6(c)), we see that LE performs better, however when we increase the number of hyper-peer clusters k_{sp}, GE becomes the prevailing approach. Notice that the real datasets performance is similar to the uniform datasets, since the data objects are uniformly distributed among the peers. In all cases, the radius of our best estimation overestimates – even slightly – the real k-th NN radius that ensures us that all similar objects are retrieved, while the area searched during the query processing increases slightly compared to the query with the real k-th NN radius.

In Fig. 6(d) and Fig. 6(e), we show the performance of nearest neighbor search on the VEC and the Covtype datasets, in terms of traffic volume. We observe that for GE the volume decreases significantly with increasing k_{sp} values, while for LE it remains practically stable. LE is not effected by the k_{sp} since the radius estimation is performed with respect to only one super-peer and its clusters. On the other hand, GE is calculated based on the histograms that are attached on the super-peer. As the k_{sp} is higher, the overlap of the clusters is reduced. Thus, each super-peer has a more detailed view of the clusters, leading to a more accurate estimation. Finally, in Fig. 6(f), we show the processing time for the Covtype dataset.

9. CONCLUSIONS AND FUTURE WORK

In this paper we presented *SIMPEER*, a novel framework for similarity search in P2P networks. Our framework is based on a novel three-level clustering scheme utilizing a set of distributed statistics, in the form of histograms. These statistics are maintained by the super-peers, in order to estimate the radius of the k-th nearest neighbor distance. We provide algorithms for efficient distributed range query processing, while nearest neighbor queries are mapped into range queries according to the estimated radius. Finally, our experimental evaluation employs real and synthetic data collections and shows that our approach performs efficiently

both in terms of computational and communication costs.

10. ACKNOWLEDGMENTS

The authors would like to thank Kjetil Nørvåg for helpful discussions on the topics of the paper, and David Novak and Pavel Zezula for providing access to the VEC dataset.

11. REFERENCES

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *SIGMOD Conference*, pages 181–192, 1999.

[2] 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'04*, pages 304–313, 2004.

[3] M. Bawa, T. Condie, and P. Ganesan. LSH forest: self-tuning indexes for similarity search. In *Proceedings of WWW'05*, pages 651–660, 2005.

[4] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *Proceedings of SIGCOMM'04*, pages 353–366, 2004.

[5] I. Bhattacharya, S. R. Kashyap, and S. Parthasarathy. Similarity searching in peer-to-peer databases. In *Proceedings of ICDCS'05*, pages 329–338, 2005.

[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. A cost model for similarity queries in metric spaces. In *Proceedings of PODS'98*, pages 59–68, 1998.

[8] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. P-tree: a P2P index for resource discovery applications. In *Proceedings of WWW'04*, 2004.

[9] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of ICDCS'02*, page 23, 2002.

[10] C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis. DESENT: Decentralized and distributed semantic overlay generation in P2P networks. *IEEE Journal on Selected Areas in Communications (J-SAC)*, 25(1):25–34, 2007.

[11] F. Falchi, C. Gennaro, and P. Zezula. A content-addressable network for similarity search in metric spaces. In *Proceedings of DBISP2P'05*, pages 126–137, 2005.

[12] 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'04*, pages 444–455, 2004.

[13] A. Gupta, D. Agrawal, and A. E. Abbadi. Approximate range selection queries in peer-to-peer systems. In *Proceedings of CIDR'03*, 2003.

[14] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, 28(4):517–580, 2003.

[15] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, pages 275–286, 1998.

[16] 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.

[17] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. Baton: a balanced tree structure for peer-to-peer networks. In *Proceedings of VLDB '05*, pages 661–672, 2005.

[18] 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'06*, page 34, 2006.

[19] 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.

[20] A. K.H.Tung, R. Zhangz, N. Koudas, and B. C. Ooi. Similarity search: A matching based approach. In *Proceedings of VLDB'06*, pages 631–642, 2006.

[21] V. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, 1(8), 1965.

[22] B. Liu, W.-C. Lee, and D. L. Lee. Supporting complex multi-dimensional queries in P2P systems. In *Proceedings of ICDCS'05*, pages 155–164, 2005.

[23] D. Novak and P. Zezula. M-Chord: a scalable distributed similarity search structure. In *Proceedings of InfoScale'06*, page 19, 2006.

[24] N. Ntarmos, T. Pitoura, and P. Triantafillou. Range query optimization leveraging peer heterogeneity. In *Proceedings of DBISP2P'05*, 2005.

[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of SIGCOMM'01*, pages 161–172, 2001.

[26] O. D. Sahin, F. Emekçi, D. Agrawal, and A. E. Abbadi. Content-based similarity search over peer-to-peer systems. In *Proceedings of DBISP2P'04*, pages 61–78, 2004.

[27] O. D. Sahin, A. Gupta, D. Agrawal, and A. E. Abbadi. A peer-to-peer framework for caching range queries. In *Proceedings of ICDE'04*, page 165, 2004.

[28] T. Seidl and H.-P. Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *Proceedings of VLDB'97*, pages 506–515, 1997.

[29] 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'05*, pages 173–180, 2005.

[30] 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'01*, pages 149–160, 2001.

[31] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis. SKYPEER: Efficient subspace skyline computation over distributed data. In *Proceedings of ICDE'07*, 2007.

[32] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of ICDE'03*, pages 49–, 2003.

[33] 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'01*, 2001.