On Efficient Top-k Query Processing in Highly Distributed Environments

Akrivi Vlachou[1] *, Christos Doulkeridis[1], Kjetil Nørvåg[2], Michalis Vazirgiannis[1]
[1]Department of Informatics, Athens University of Economics and Business, Greece
[2]Department of Computer Science, NTNU, Trondheim, Norway
{avlachou,cdoulk,mvazirg}@aueb.gr, Kjetil.Norvag@idi.ntnu.no

ABSTRACT

Lately the advances in centralized database management systems show a trend towards supporting rank-aware query operators, like top-k, that enable users to retrieve only the most interesting data objects. A challenging problem is to support rank-aware queries in highly distributed environments. In this paper, we present a novel approach, called SPEERTO, for top-k query processing in large-scale peer-to-peer networks, where the dataset is horizontally distributed over the peers. Towards this goal, we explore the applicability of the skyline operator for efficiently routing top-k queries in a large super-peer network. Relying on a thresholding scheme, SPEERTO returns the exact results progressively to the user, while the number of queried super-peers and transferred data is minimized. Finally, we propose different variations of SPEERTO that allow balancing between transferred data volume and response time. Through simulations we demonstrate the feasibility of our approach.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

Algorithms, Experimentation, Performance

Keywords

Top-k queries, skyline operator, peer-to-peer systems

1. INTRODUCTION

Recently there has been an increased interest in database management systems to incorporate and support more flexible query operators, like top-k, that help in avoiding huge

and overwhelming result sets. Top-k queries retrieve the objects that best match the user requirements by employing user-specified scoring functions that result in an ordered set of objects containing the best k objects only [8, 14].

A number of applications can significantly benefit from support for top-k query processing, for example multimedia retrieval (including images) [9, 12], digital libraries [15, 16], web search [18], and e-commerce [17]. Consider for example online booking systems, e.g, travel and accommodation, where the user is only interested in the best offers (airtickets, hotels) according to a set of dynamic, user-specified criteria.

Due to applications and systems such as sensor networks, data streams, and peer-to-peer (P2P) systems, data generation and storage is becoming increasingly distributed. Thus an emerging challenge is to support top-k query processing over a highly distributed network of collaborative computers (i.e. servers or peers). In this paper we focus on top-k query processing in P2P systems, which is a context that can also easily be generalized to other distributed systems. There is only limited previous work on supporting top-k queries in P2P systems, and those approaches either assume vertical data partitioning to peers [18], rely on caching techniques [4, 22], or deliver approximate query result sets [13]. In contrast to these approaches, our work assumes horizontal data partitioning among peers and supports a large class of scoring functions. Each user may define his/her own arbitrary preferences for each query, therefore the queries are not necessarily re-occurring, which renders caching techniques inappropriate. The challenge is to provide efficient algorithms for processing top-k queries, i.e. queries that return only the exact best k results to the user.

In this paper we present SPEERTO[1], a framework that supports top-k query processing over horizontally partitioned data stored on peers organized in a super-peer network. Users are allowed to specify a monotone function for each query that aggregates a certain number of the objects characteristics into a single *score* that defines a total ordering, and therefore enables the retrieval of top-k results. Our approach is based on novel use of the skyline operator [5] for answering top-k queries. For a maximum value of K, denoting an upper bound on the number of results requested by any top-k query ($k \leq K$), each peer computes its K-skyband [20] as a pre-processing step. Each super-peer maintains and

aggregates the K-skyband sets of its peers to answer any incoming top-k query. By exchanging skyline sets (which are a subset of the K-skyband sets) at super-peer level, SPEERTO always provides the exact and complete result set in a progressive way, while queries are deliberately routed to those super-peers that actually contribute to the top-k ($k \leq K$) result.

To summarize, SPEERTO utilizes a threshold-based super-peer selection mechanism based on the skyline points of each super-peer. Although the skyline operator [5] has received recently considerable attention, the usage of the skyline operator for answering top-k queries has yet not been explored. We study the correctness of our approach and its effectiveness in terms of number of queried super-peers and transferred objects. The main contributions of our work are:

- An exploration of the applicability of the skyline operator for efficiently answering top-k queries for a wide class of scoring functions, indicating user-specified preferences, in large P2P networks.

- A presentation of SPEERTO, a novel framework which efficiently supports progressive processing of top-k queries using the skyline and K-skyband sets, employing a thresholding scheme in order to facilitate pruning of objects that cannot belong to the result set. Moreover, the correctness and optimality in terms of queried super-peers and transferred data are demonstrated.

- A detailed study of different variations of SPEERTO in order to balance between different performance metrics, like response time and volume of exchanged data.

- An extensive experimental evaluation showing that our approach performs efficiently and provides a viable solution. We also evaluate SPEERTO for top-k queries with $k > K$ where we achieve high recall values.

The rest of this paper is organized as follows: Section 2 reviews the related work, and in Section 3 we provide the preliminaries for presenting SPEERTO. In Section 4, we describe how to construct the skyline-based routing mechanism for top-k query processing over a super-peer architecture. Thereafter, in Section 5 our threshold-based top-k algorithm is presented, while in Section 6 we discuss extensions of SPEERTO. The experimental evaluation is presented in Section 7, and finally we conclude in Section 8.

2. RELATED WORK

Several papers have dealt with the issue of top-k query processing in centralized database management systems [8, 14]. Previous work in distributed environments [9, 12, 17] has focused on vertically distributed data over multiple sources, where each source provides a ranking over some attributes. Most approaches, such as recently [2], try to improve some limitations of the Threshold Algorithm [11]. A common underlying assumption of these papers is that data is vertically distributed to nodes, in contrast to our case where we assume horizontal distribution of data. Marian *et al.* [17] study top-k query evaluation over web-accessible databases, including random accesses to score lists, instead of sorted accesses only, as in [11]. Following the same concept, there exists some previous work for top-k queries in P2P over vertically distributed data. In [6], Cao and Wang

propose an algorithm called "Three-Phase Uniform Threshold" (TPUT) that aims to prune unnecessary data objects and it is guaranteed to terminate in three round-trips. Later, TPUT was improved by KLEE [18]. KLEE has two variants, one that requires three phases and another that only needs two round-trips. KLEE also provides mechanisms for trading performance with result quality, thus supporting approximate top-k retrieval. However, processing top-k queries in the context of horizontally distributed data and P2P systems has not been adequately addressed yet.

For horizontally distributed data among peers, P2P top-k query processing has been studied in only a few works so far. Balke *et al.* [4] try to minimize the data object traffic induced by top-k processing. However, this approach requires that each query is processed by all super-peers, unless the exact same query reoccurs, which is unlikely as there is an infinite number of potential queries posed by different users. A similar approach for unstructured P2P systems is presented in [1], where the main technique is a variant of flooding, followed by a merging score-list step at intermediate peers. In [22], the authors rely on result caching to prune network paths and answer queries without contacting all peers. Their approach relies on caching techniques, therefore the performance is dependent on the query distribution. Even more important, they assume acyclic networks, which is restrictive for dynamic peer-to-peer networks. Hose *et al.* [13] construct routing filters in the form of histograms, in order to prune query paths and return approximate results. These filters are built on each peer progressively, as the peer communicates with other peers, using a query feedback approach. However this approach delivers approximate answers and the performance drops with increasing dimensionality since multi-dimensional histograms should be used.

In the area of P2P information retrieval, there exists some work that takes into account top-k queries. However this work is not entirely within the context of our work, as their main focus is on document retrieval and on defining an appropriate scoring function. For example, in [15, 16], Lu and Callan focus on search in a digital library context, using hierarchical P2P networks and propose result merging algorithms based on sampled documents from neighboring peers.

Finally, the skyline operator [5] has recently received considerable attention, but its usage for answering top-k queries has not been explored yet. In [21] the authors improve the performance of ranked join indices based on the concept of dominating sets. In [19] a method for continuous top-k queries over streams is presented that monitors the top-k objects by using the K-skyband on a two dimensional transformed score-time space.

3. PRELIMINARIES

In this section, we present the problem statement, the basics regarding top-k queries, and a short overview of the P2P system where the proposed approach is deployed. An overview of the symbols used can be found in Table 1.

3.1 Problem Statement

Given a data collection O of n objects o_i ($1 \leq i \leq n$), we assume d features $s_j(o_i)$ ($1 \leq j \leq d$) that describe an object $o_i \in O$. We assume that the features s_j are numerical scoring functions with non-negative values that evaluate certain features of database objects. For example, s_j can be extracted characteristics, aggregations of attribute values,

Symbols	Description
d	Data dimensionality
n	Dataset cardinality
K	The maximum number of k
N_p	Number of peers
N_{sp}	Number of super-peers
DEG_p	Degree of simple peer
DEG_{sp}	Degree of super-peer
SKY_i	Skyline set of the ith super-peer
$KSKY_i$	K-skyband set of the ith super-peer

Table 1: Overview of symbols

Figure 1: Feature space

or scoring functions for low-level features [3]. Furthermore, without loss of generality, we assume that smaller score values are preferable.

The feature space is defined by the d scoring functions s_j, therefore it is a d-dimensional space. An object $o_i \in O$ can be represented as a point p in the feature space: $p = \{p[1], ..., p[d]\}$, where $p[j] = s_j(o_i)$ is a value on dimension d_j. Figure 1 depicts a 2-dimensional example. In the rest of this paper we use the terms object and data point interchangeably.

In our approach we assume an aggregation function f that is increasingly monotone, i.e. if $p[i] \leq p'[i]$ for every i, then $f(p) = f(p[1], ..., p[d]) \leq f(p'[1], ..., p'[d]) = f(p')$. The restriction of monotonicity is a common property [8, 11] and it conveys the meaning that whenever the score of all dimensions of the point p is at least as good as another point p', then we expect that the overall score of p is as least good as p'. The result of a top-k query is the ranked list of the k objects with lowest *score* values.

A special case of monotone functions is the weighted sum function, also called linear. Each feature $s_j(o_i)$ has an associated query-dependent weight w_j indicating s_j's relative importance for the query. The aggregated score for object o_i is defined as a weighted sum of the individual scores: $score(o_i) = \sum_{j=1}^{d} w_j \times s_j(o_i)$, where $w_j \geq 0$ ($1 \leq j \leq d$) and $\exists j$ such that $w_j > 0$. As some weights can be set equal to zero, our approach also supports top-k queries with respect to only to a subset of the available features. The weights indicate the user's preferences and influence the ordering of the data objects and therefore the top-k result set. Consider for example the dataset depicted in Figure 1. By assigning a high weight to feature s_2, point p is the top-1 object, while if a low weight is used, point q becomes the top-1 object.

Our approach is applicable for any increasingly monotone aggregate function, but in our examples we use the weighted sum function, which is one of the most common scoring functions for top-k retrieval. In our setting a top-k query $q_k(f)$ takes two parameters: a user specified monotone function f and the number of requested objects k. In the special case of the weighted sum, the user specifies the weighting of each feature, i.e. how important this feature is based on his preferences. Therefore the query can be expressed as $q_k(w)$, where w is a d-dimensional vector $w = \{w_1, ...w_d\}$. Notice that both the scoring function and the parameter k may differ for each query and we are interested in retrieving the k objects with the best (minimum) values of the scoring function.
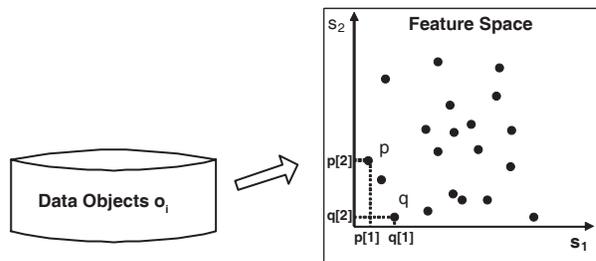
3.2 System Overview

The overall aim is to provide a routing mechanism for answering top-k queries in P2P networks, assuming a super-peer architecture. More formally, we assume an unstructured 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 ($1 \leq i \leq N_{sp}$), 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 super-peer maintains links to simple peers, based on the value of its degree parameter DEG_p. In addition, a super-peer is initially connected to a limited set of at most DEG_{sp} other super-peers ($DEG_{sp} < DEG_p$). Later, at query time, each super-peer is able to open direct connection to any other super-peer in the network, using its IP address. However notice that the approach is also applicable, when no direct connection between super-peers can be established, and the communication is achieved by query forwarding through other super-peers.

Each peer P_i holds n_i d-dimensional points, denoted as a set O_i ($1 \leq i \leq N_p$). Since we assume horizontal data distribution, the size of the complete set of points is $n = \sum_{i=1}^{N_p} n_i$ and the dataset O is the union of all peers' datasets O_i ($O = \cup O_i$). Each peer maintains its own data objects, such as images or documents, and only the feature values of few selected objects, namely the K-skyband [20] points, are published as representative points to the respective super-peer, while the original data is stored at the peer. By maintaining the K-skyband points, any super-peer is capable of answering any top-k ($k \leq K$) query as far as only the data of all peers connected to the super-peer are concerned. The remaining challenge is to answer top-k queries over the entire super-peer network, in a way that only super-peers that contributed to the query are contacted. In the following, we propose an approach that supports top-k queries over data distributed in a super-peer network, utilizing routing indices based on skylines [5]. Our technique guarantees accurate results, while minimizing the number of queried super-peers and the amount of network traffic.

4. SKYLINE-BASED ROUTING

In this section, we first discuss the relation between top-k and skyline queries (Section 4.1). Thereafter, in Section 4.2, we describe how to construct the skyline-based routing mechanism for top-k query processing over a super-peer architecture. Finally, in Section 4.3, we outline the effects of churn and dynamic data on the routing mechanism.
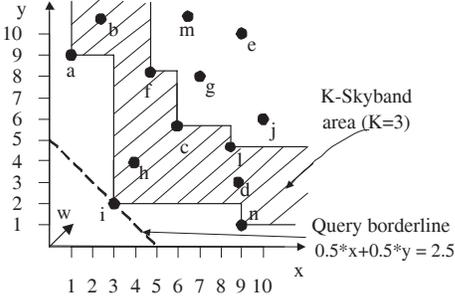
Figure 2: Skyline, K-skyband, and top-k

4.1 Top-k and Skyline Queries

Consider a linear top-k query defined by a vector w. As discussed in [21] the magnitude of the query vector does not influence the query result as long as the direction remains the same. Therefore, we make the assumption that $\sum_{1 \leq i \leq d} w_i = 1$. In a d-dimensional space we define as *query borderline* the $d-1$ dimensional hyper-plane, which is vertical to vector w and contains the furthest data point, which belongs to the top-k result set. If we consider the 2-dimensional example depicted in Figure 2 the point i is the top-1 object for the query $0.5 * x + 0.5 * y$. The line $0.5 * x + 0.5 * y = f(i) = 2.5$ is considered as the borderline for the top-1 query with $w = [0.5, 0.5]$. In the following we define the skyline set and discuss its relation to top-k queries.

Skyline Definition. Assuming a space D defined by d dimensions $\{d_1, d_2, .., d_d\}$ and given a set of points O, a point $p \in O$ with $p = \{p[1], ..., p[d]\}$ is said to *dominate* another point $q \in O$, if on each dimension $d_i \in D$, $p[i] \leq q[i]$; and on at least one dimension $d_j \in D$, $p[j] < q[j]$. The *skyline* is a set of points $SKY \subseteq O$ which are not dominated by any other point. The points in SKY are called *skyline points*.

In the example of Figure 2, a, i and n are the skyline points. Notice that i, which is the best match for the top-1 query with $w = [0.5, 0.5]$, belongs to the skyline.

Observation. The top-1 object for any increasingly monotone function belongs to the skyline set.

Proof: Consider a point q that does not belong to the skyline, but it is the top-1 for a query defined by an increasingly monotone function f. Then there exists another point p that dominates q, i.e. on each dimension $d_i \in D$, $p[i] \leq q[i]$; and on at least one dimension $d_j \in D$, $p[j] < q[j]$, and since f is increasingly monotone this leads to a contradiction, because q is the top-1, i.e. $f(p) > f(q)$. Thus, the top-1 object for any increasingly monotone function belongs to the skyline.□

Motivated by the fact that the top-1 always belongs to the skyline for any monotone function, we use the skyline as a pre-processing step to answer top-k queries. In order to collect the points necessary to answer exact top-k queries, we adopt the concept of K-skyband [20]. A K-skyband query returns the set of points which are dominated by at most $K-1$ other ones. Thus, the conventional skyline is a special instance of the K-skyband, where $K = 1$. In Figure 2, the K-skyband for $K = 3$ includes all points that lie in the line-shadowed area. Notice that this area contains the top-3 points for any query.

4.2 Routing Mechanism Construction Phase

In SPEERTO the super-peers are responsible for answering arbitrary top-k queries over their peers' data. This is enabled by a *pre-processing* phase where super-peers gather some carefully selected data from their peers. Thereafter, at query processing time, a super-peer can execute the query over its locally aggregated data and retrieve the fraction of the top-k query result that corresponds to its peers.

As discussed before, the result of top-k queries for any increasingly monotone function can be answered from the K-skyband (where $k \leq K$). The K-skyband is a set of points, such that there exists no other point that can belong to the result of any top-k query for any increasingly monotone function. In our approach each peer computes its K-skyband during a construction phase. Each super-peer gathers K-skyband sets from its simple peers and merges the individual K-skyband sets by discarding points that are dominated by more than $K-1$ points. In this way a super-peer is capable to answer any incoming top-k query over its peers' data. The choice of the algorithm used by the peer for the K-skyband computation is indifferent to our framework, as it does not influence its performance. It should be stressed that even though the skyline operator is CPU-intensive [7] and therefore more costly than a top-k query, SPEERTO uses the skyline as a pre-processing step, i.e. its construction is a one-time cost, and then any top-k query with arbitrary k ($k \leq K$) and scoring function can be processed (see Section 5).

Given the K-skyband at each super-peer, there exist two naive solutions to process global top-k queries. In the first, each super-peer broadcasts its K-skyband to all other super-peers, then each super-peer has enough data to answer any top-k ($k \leq K$) query locally. The advantage is that the query is processed (at any super-peer) without contacting remote super-peers. However, this approach is not feasible in a highly distributed environment, because of the size of the skyband and the cost of distributing it to all the super-peers and keeping it updated.

The second naive approach is to flood each query to all super-peers to find the correct top-k result. The advantage of this approach is that the cost of distributing the skybands is avoided. However, flooding is costly, and although appropriate for distributing metadata needed for creating routing indices, it is too costly to employ for each individual query.

SPEERTO employs a more efficient approach than the naive approaches described above, that combines the advantages of the aforementioned approaches, while alleviating the disadvantages. SPEERTO broadcasts only some summary information of the K-skyband, namely the skyline set, and utilizes a threshold-based super-peer selection mechanism (see Section 5). Intuitively, the skyline is the border of the K-skyband with respect to the axes. Notice that the cardinality of the skyline is significantly smaller than the cardinality of the K-skyband.

4.3 Routing Indices, Updates and Churn

Routing indices at super-peer level are built during the construction phase, prior to query processing. A super-peer first assembles the K-skyband of its peers, then computes a new K-skyband over the assembled peer data, and finally it computes its skyline. The skyline information is broadcast to other super-peers and serves as routing index. Essentially, in this way, each super-peer receives and maintains the sky-

line of other super-peers. Our approach for top-k queries in a P2P system gives correct and complete results under the assumption of a system with no churn and static data. In this section, we discuss about churn and dynamic data.

Data updates only infrequently change the skyline, and small changes in the skyline do not significantly change the accuracy of the top-k query processing (see Section 7). Therefore, it is not necessary to continuously maintain the skyline updated at remote super-peers, and periodic updates suffice. Obviously, high update rates can lead to time intervals where the results may not be accurate temporarily. The maintenance approach of remote skylines is based on broadcasting the skyline updates, when either the skyline has significantly changed or the validity time has expired. While skylines are used to select super-peers during top-k processing, the K-skyband on a super-peer is used to generate the actual results, and has to be more frequently updated. However, this cost is still less significant because a peer is relatively close to its super-peer in terms of network distance. In Section 6, we discuss an extension of SPEERTO which is more robust to updates.

Churn of super-peers is detected by lack of response during query processing. When this occurs, the skyline entry of the departed super-peer is removed at the querying super-peer. When a super-peer joins the network, its skyline is broadcast as described previously. Churn of simple peers is handled by recomputing the super-peer K-skyband, when a simple peer leaves or joins. In the case that the skyline of the super-peer is modified, the skyline has to be updated at all super-peers, in order to ensure accurate results.

5. P2P TOP-K QUERY PROCESSING

In this section, we introduce a threshold-based algorithm (Section 5.1) that answers arbitrary top-k queries by querying only the necessary super-peers. In the following, we also show that our threshold-based algorithm is optimal in terms of the number of contacted super-peers and the volume of transferred data (Section 5.2).

5.1 Threshold-based Top-k Algorithm

Our distributed top-k algorithm assumes that there exists a construction phase, where each peer computes the K-skyband of its local data. Then, each super-peer gathers the K-skyband sets of all its associated peers. The K-skyband information is merged at each super-peer, resulting in a new K-skyband set, denoted as $KSKY_i$, which is stored locally at the super-peer. Finally, each super-peer broadcasts to the other super-peers only some summary information of the K-skyband, namely the skyline set, sufficient for every super-peer to route any top-k query to those super-peers that can contribute to the final result. Each super-peer SP_i assembles N_{sp} sets of skyline points SKY_i, ($1 \le i \le N_{sp}$). These points are called *routing objects*.

Actually, the super-peers store the *feature values* of the objects that are stored on the peers. During query processing the super-peer retrieves the entire *data object* from the corresponding peer. Note that in contrast to routing objects that only contain the IP address of their super-peer, data objects may contain more information (and hence be much larger) than the routing objects[2]. During query processing

---

[2]An example is a data object that is an image, the routing object can in this case be just descriptive features.
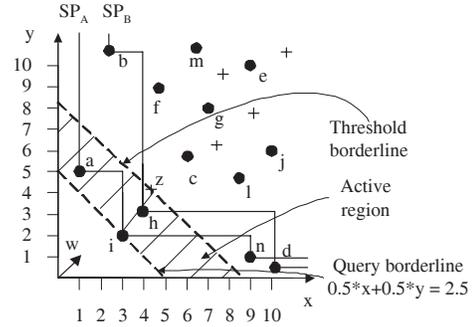
Figure 3: Example of query borderline

data objects that probably belong to the query result set are also retrieved from peers indexed by other super-peers and transferred through these super-peers to the querying super-peer. For sake of simplicity, we refer to the points that belong to the K-skyband of a super-peer also as data objects, and ignore the fact that the actual data objects are transferred at query time from the peer to the super-peer. The communication cost between super-peer and peer is considered as less significant than the query propagation through the super-peer network because a peer is relatively close to its super-peer in terms of network distance. In case that the data objects do not contain more information than the feature values, there is no communication between super-peer and peer during query processing.

In the following, we describe our threshold-based top-k algorithm assuming that every super-peer SP_i ($1 \le i \le N_{sp}$) stores the routing objects SKY_i of all other super-peers ($1 \le i \le N_{sp}$) and the merged K-skyband set ($KSKY_i$) of its associated peers only.

Let us first consider a two dimensional space as a showcase scenario as illustrated in Figure 3. In the figure, we depict the routing and data objects stored at super-peer SP_A. In more details, the skylines of two super-peers SP_A and SP_B are shown along with the routing objects from the other super-peers. For clarity reasons, we label only the routing objects of the two super-peers SP_A and SP_B that are actually involved in the top-k query of the example. Routing objects are depicted as black circles, while data objects are marked with crosses. The data objects are the points that belong to the $KSKY_A$, i.e. the aggregated K-skyband set of super-peer SP_A. Therefore, Figure 3 depicts the information that is available to super-peer SP_A when query processing starts. As described in more detail below, during query processing more data objects are transferred to the querying super-peer SP_A through the neighboring super-peers.

In a two dimensional space, progressive processing a top-k query given an arbitrary weighting is similar to sweeping the query borderline, with specific slope defined by the query weights, through space from the axes toward the data. The first data point that the line meets is the top-1, the second the top-2, etc. until it finds top-k data points. Actually, each time the line meets a data point, this point can be immediately returned to the user, as it is really the next top object of the query (progressive property of our algorithm). Note that some of the points that the borderline meets are routing objects. In this case the routing object must be

Algorithm 1 Query processing on super-peer SP_Q

1: **Input:** Query $q_k(f)$
2: $list = \{\emptyset\}$
3: $list = SP_Q.\text{query}_{\cup SKY_i}(q_k(f))$
4: $threshold = f(list[k])$
5: $c = 0$
6: **while** $(c < k)$ **do**
7: $next\_obj = list.\text{pop}()$
8: **if** $next\_obj$ is a *routing object* **then**
9:   $SP = next\_obj.\text{super-peer}()$
10:   $temp = SP.\text{query}(q_{k-c}(f), threshold)$
11:   $list.\text{removeRoutObj}(SP)$
12:   $list.\text{add}(temp)$
13: **else**
14:   return $next\_object$ to the user
15:   $c = c + 1$
16: **end if**
17: $threshold = f(list[k - c])$
18: **end while**

Figure 4: Example of top-k algorithm

higher score cannot belong to the final result set. In each iteration the top object of the list is examined. Then a top-k query is broadcast to the super-peer (SP) responsible for this object. After SP's data objects are retrieved by SP_Q, all routing objects of SP are removed from the sorted list before inserting its data objects, since they are no longer necessary to maintain. Then, the threshold is updated with the scoring value of the k-th object in the list. In each subsequent iteration, if a data object is retrieved, it is returned to the user as the top-1, top-2, etc. result. Otherwise, if a routing object is retrieved, a top-$(k - c)$ query is send to the corresponding super-peer along with the current threshold value (c denotes the number of results returned thus far to the user). The super-peer sends back $k - c$ objects, or less if there are not $k - c$ objects with value below the threshold. The algorithm terminates when k data objects have been retrieved from the sorted list.

Example: Consider a small super-peer network consisting of four super-peers $SP_A, \ldots SP_D$, and a querying super-peer $SP_Q = SP_A$ that has assembled the skylines (routing objects) of the other super-peers, as depicted in Figure 4. Let us assume that SP_A needs to answer a top-3 query with a linear aggregate function that assigns equal weights to both dimensions. On the right side of the figure, the skyband information maintained on each super-peer is depicted in tables. The grey-shadowed objects are the skyline objects that are broadcast to other super-peers and they are also depicted on the left part of the figure graphically. According to Algorithm 1, the sorted list is initialized with routing objects: $i(3,2)$, $m(6,0.5)$ and $h(4,3)$ and threshold is set to 3.5. The first object that is processed is object $i(3,2)$ that belongs to super-peer SP_B. Thus SP_A sends a top-3 query to SP_B, and retrieves its local (at SP_B) top-3 results. These data objects are $i(3,2)$, $(4,2.5)$ and then for the third ranked object there are actually three objects with the same aggregate score $o(9,1)$, $a(1,9)$ and $(7,3)$. These three data objects are not returned to SP_A since they are discarded by the threshold value. So, only two points are returned to SP_A by SP_B and they are merged with the objects already existing in the list. The threshold value is set to 3.25, as the new k-th object $(4,2.5)$ has a lower score value than the old threshold value. Then $i(3,2)$ is returned to the user as top-1. Therefore

replaced by some data points of the super-peer to which the routing object belongs to. At each step the query borderline is an indication of how far we have examined the data space and it guarantees that there does not exist any other point in the examined space that has not been retrieved yet. This guarantees that there is no data point that has a smaller scoring value than the retrieved points.

A threshold value is defined as the score of the k-th routing or data object encountered so far. In the 2-d space, this defines a *threshold line* that gradually sweeps the space towards the axes origin. The region defined between the query and the threshold borderline is called *active region* and it contains at least $(k - c)$ objects, where c is the number of data points that is already returned to the user. In each step, the active region contains all objects that may appear in the final result set. Notice that the querying super-peer is not aware of all data points that fall in the active region, therefore if a routing object is retrieved, the query must be broadcast to the corresponding super-peer.

Continuing the example depicted in Figure 3, consider a linear top-4 query with weights $w = (0.5, 0.5)$. Let us further assume that the query is posed at super-peer SP_A. The top-4 objects (i, a, h and z) based on the data and routing objects stored on SP_A are retrieved, and the score of the 4th object (z), defines the threshold borderline. This guarantees that the results of this top-k query are found in the active region. Notice that some data points of SP_B may fall in the active region and therefore point z may not belong to the top-4 result set. First, the routing object i is examined and since it belongs to SP_A the data object i is retrieved and returned to the user. In the next step, point a is retrieved and returned as the top-2 point. Afterwards, we retrieve the routing object h that belongs to SP_B. Therefore, super-peer SP_B is queried and assuming that no other data point of SP_B falls in the active region, points h and z are returned to the user.

Algorithm 1 describes how P2P top-k query processing is performed. The routing and data objects retrieved thus far are kept in a sorted list based on the scoring value. This list is initialized by the querying super-peer (SP_Q) with the top-k objects of the skyline results $\cup SKY_i$. We use as threshold the scoring value of the k-th object, as any other object with

the list becomes of size 2 and it contains: $m(6, 0.5)$, $(4, 2.5)$. Next m is processed and as it belongs to SP_C, SP_A sends a message to SP_C requesting its top-2 objects. SP_C returns $m(6, 0.5)$, while $f(2,6)$ is pruned by the threshold. Thereafter, m is returned to the user and the list contains only one object $(4, 2.5)$. This object is processed next, and since it is a data object, it is returned to the user immediately as the top-3. Finally, the algorithm terminates.

5.2 Correctness and Optimality

The usage of the threshold ensures that SPEERTO progressively returns accurate and exact answers for any top-k ($k \leq K$) query. Moreover, it reduces communication costs by preventing unnecessary data objects from being transferred in the network during query processing. SPEERTO also avoids querying super-peers that do not contribute to the result set. In the following we assume that the query is answered using a snapshot of the P2P network, i.e. static network and contents, and we show the correctness of our algorithm and the optimality in terms of queried super-peers and transferred data.

Correctness of the algorithm: After k data objects have been examined, an object that has not been seen cannot be in the top-k result set. Observation 1 ensures that for every super-peer SP and for any increasingly monotone function the best match for SP is a routing object that exists on SP_Q. The objects are kept sorted by the scoring function of the query, thus, if the next object is a data object o this means that there is no super-peer for which its best match has a better score than o. Otherwise, if the next object is a routing object, the corresponding super-peer is queried. Therefore, all super-peers that may contribute to the result set are queried. Finally, the objects that are discarded due to the threshold cannot belong to the result set since there exist at least k objects with a better score. So, our algorithm does not suffer by false negatives or positives.

Minimized number of queried super-peers: A super-peer SP is queried only if a routing object o is the next best match in the sorted list. Since all points with a smaller scoring value have been examined previously, the routing object o corresponds to the next best match. So, there is at least one point of SP that contributes to the top-k result set, namely o, and thus avoiding to query SP would lead to a wrong result set.

Minimized transferred data during query processing: Let o be an object that has a score smaller than the threshold and can be discarded without violating the accuracy of our algorithm. Since the threshold is larger than the score of o, there are less than k objects in the sorted list with a smaller score. If all objects in the sorted list that have a smaller score than object o are data objects, then o belongs to the top-k result set. This leads to a contradiction, since the accuracy of the algorithm is violated. Therefore any object with smaller value than the threshold cannot be discarded, and our algorithm transfers the minimum number of data objects during query processing.

To summarize the benefit of the proposed threshold-based algorithm is threefold: a) results can be returned progressively to the user, b) communication costs are reduced, by defining a threshold value, which prevents unnecessary data objects to be transferred in the network during query processing, and c) the number of contacted super-peers is minimized during query processing.

6. EXTENSIONS OF SPEERTO

In this section we study some extensions of the SPEERTO framework. By relaxing SPEERTO's optimality in terms of number of queried super-peers and transferred objects, we manage to a) decrease the response time and b) decrease storage and – more importantly – maintenance costs due to updates. In order to reduce the response time, we propose a variant of SPEERTO that queries in parallel more than one super-peer (Section 6.1). Thereafter, we propose an extension that restricts the cardinality of the skyline (Section 6.2) aiming to reduce the construction and maintenance costs of the SPEERTO framework.

6.1 Parallel Query Processing

In each iteration, Algorithm 1 examines the first object in the sorted list and queries the corresponding super-peer. A straightforward extension of this approach is to query more than one super-peers in each iteration simultaneously, using the current threshold. In the original algorithm, by querying each super-peer and then adjusting (when possible) the threshold, the query is sent to the next super-peer, only after the previous one has returned its results. In the parallel variant, the querying part is non blocking, therefore the total response time is reduced. On the other hand we cannot guarantee the optimality of the amount of transferred data, since some super-peers are queried using a higher threshold, than in the case of refining the threshold after each super-peer was queried. Moreover, some super-peers that do not contribute to the final result set may be queried, while they could have been discarded by data objects retrieved by previously queried super-peers.

The remaining question is how many super-peers should be queried in each iteration. The most simple way is to have a fixed number, according to the querying super-peer's SP_Q traffic/workload and maximum number of open connections that can be established. However this approach adapts neither to the number of queried objects k, nor to the number of objects that have already been retrieved. Therefore, it is more effective to dynamically estimate the number of necessary super-peers to be queried, in order to retrieve the top-k objects. Each super-peer may employ its own heuristic based on statistics. In the following, we present a simple strategy that does not rely on any statistic or on previous queries.

The parallel variant works in the following way. Initially, one super-peer is queried using $q_k(f)$. All N_r retrieved objects have a score less than the current threshold t and higher than the score of the first retrieved object o_1, i.e. $score(o_1)$. So we estimate the mean score from this super-peer as: $m = \frac{t - score(o_1)}{N_r}$. The next object ($o_2$) in the list is examined and if it is a routing object we estimate the number of objects that will be retrieved: $\overline{N_{r2}} = \frac{score(o_2) - score(o_1)}{m}$. Then the next object is examined. If it is a data object, the super-peers that are found so far, are queried. If it is a routing object (o_3), we estimate again the number of objects $\overline{N_{r3}} = \frac{score(o_3) - score(o_2)}{m}$ that will be retrieved. This continues until either a data object is found, or so many routing objects have been examined that the estimated number of objects ($\overline{N_{r2}} + \overline{N_{r3}}$) is more than the $(k - c)$ remaining objects to be retrieved. After the super-peers found so far are queried, each returns N_{r2} and N_{r3} objects respectively and a new mean score is computed for each one, i.e.
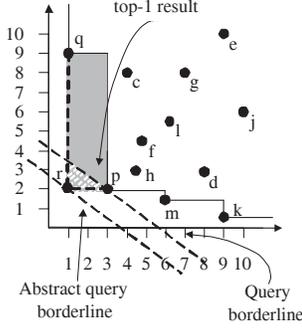
Figure 5: Abstract skyline

$m_2 = \frac{t-score(o_2)}{N_{r2}}$ and $m_3 = \frac{t-score(o_3)}{N_{r3}}$. Then, we compute a new threshold t' and the new overall mean score m' that is used instead of m as: $m' = \frac{m+m_2+m_3}{3}$. The algorithm continues until k data objects are returned to the user. It should be stressed that our approach does not require global statistics and it has no additional communication or maintenance cost.

6.2 Reducing the Skyline Cardinality

One of the major drawbacks of the skyline operator is the high cardinality $|SKY|$ of the result set [7], especially for high-dimensional or anti-correlated datasets. In our framework this would lead to high construction and maintenance cost and storage requirements at super-peers. Moreover, local query processing at each super-peer becomes more expensive for higher numbers of routing objects. Given an upper limit U, in this section we study how to *abstract* the skyline $aSKY$ with at most U points ($U < |SKY|$). This abstraction has the following properties: a) each point $p \in SKY$ is either dominated by or equal to at least one point $q \in aSKY$ b) $|aSKY| \leq U < |SKY|$, and c) it only slightly influences the routing power of the skyline, i.e. move the query borderline as little as possible. Obviously, the abstraction is not unique.

The problem is to find an approximation of the skyline of fixed size to distribute to all super-peers as a routing mechanism. The resulting trade-off is between skyline size and the accuracy of the approximation (leading to more contacted super-peers and more transferred data). Consider for example the dataset depicted on Figure 5, where the points q, p, m, k are the skyline points. Given a upper limit $U = 3$ for the skyline abstraction, let us assume that we decide to replace the points q, p with one point r. SPEERTO selects the super-peers that are contacted during query processing based on the skyline points. In order not to violate the accuracy of SPEERTO, the abstraction of SPEERTO must ensure that whenever a super-peer should be contacted based on the skyline points, the super-peer will also be contacted based on the abstraction. Let us assume that Figure 5 depicts the skyline points of one super-peer SP_A, then for any query the query borderline meets point r before points p and q. On the other hand, if a skyline point of another super-peer falls in the dashed area (triangle) then based on the abstract skyline we contact super-peer SP_A, while based on the real skyline points we could avoid contacting SP_A. Therefore, the abstraction causes an increase in the number

of contacted super-peers. In general, the larger the shadowed area in Figure 5, the higher the probability of querying more super-peers.

A benefit of the abstract skyline is that our approach becomes more robust to updates. SPEERTO can guarantee accurate query answers in the case of updates, if the routing objects of the respective skyline do not change. Otherwise, the new routing objects have to be broadcast to all super-peers. The abstract skyline is a lower bound of the skyline and is less likely to change than the original skyline. In the case of high churn rate and/or updates, the abstract skyline may be appropriate, in order to reduce the maintenance cost of routing indices and shorten the time intervals where we cannot provide exact query results. A shortcoming of the abstract skyline is that we can not use the threshold any more, since we are no longer certain that at least k objects exist in the query space.

We now present a heuristic for calculating the abstract skyline. Intuitively, each time we pick pairs of skyline points p, q that can be replaced by a new point r, until we have at most U points. Algorithm 2 describes the procedure of choosing two points and replacing them with one new point. Inspired by SFS [10], we choose the skyline point p with the largest entropy value $E(p) = \sum_{1 \leq i \leq d} \ln(p[i] + 1)$, because the smaller the entropy value the less likely p is to be dominated by other points. This insinuates that a point with lower entropy value has a stronger dominance power and therefore is considered as more important. Thereafter, we have to determine a suitable point q to merge with the selected point. For each dimension d_i we find the skyline point that has the smallest distance from p. We choose the point q that has the smallest distance in any dimension. Then p and q are replaced by point r, defined by the minimum values of p and q on all dimensions. Thereafter, all skyline points dominated by r are removed. Notice that r indeed dominates other skyline points for higher than two-dimensional spaces. This process iterates and terminates when $U < |SKY|$.

As an example, consider again the dataset depicted on Figure 5. Let us assume that point p has the largest entropy value and that point q is the nearest point based on the distance for any dimension. Then according to Algorithm 2 the two skyline points p and q are replaced with point r. This replacement does not affect the correctness of SPEERTO, since the query borderline for any query meets point r before points p and q. Therefore, the query is routed to the corresponding super-peer and the accurate top-k result set is retrieved.
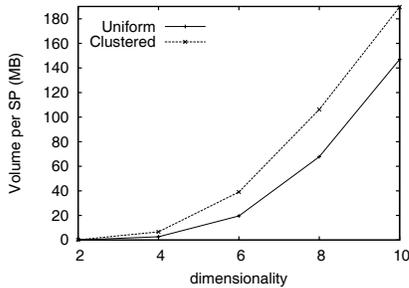
Figure 6: Construction cost

7. EXPERIMENTAL EVALUATION

We studied the performance of our framework using simulations. The simulator was implemented in Java[3] and it run on 3.8GHz Athlon Dual Core AMD processors with 2GB RAM. In order to test the algorithms with realistic network sizes, we ran multiple instances of the peers on the same machine and simulated the network interconnection.

The P2P network topology used in the experiments consists of N_{sp} interconnected super-peers in a random graph topology. In our experiments we vary the network size (N_p) from 2000 to 20000 peers. We used synthetic (uniform and clustered) data collections. The dataset is horizontally partitioned evenly among peers. The uniform dataset includes random points in a space $[0, L]^d$. For the clustered dataset, each super-peer picks cluster centroids randomly and all associated peers obtain points, the coordinates of which follow a Gaussian distribution on each axis with variance 0.025, and a mean equal to the corresponding coordinate of the centroid. We conduct experiments varying the dimensionality (2-10) and the cardinality (1M-2.5M) of the dataset. Each time we generate 20 queries with random weightings.

For all queries, we measure the average: (i) total response time (including network delay), (ii) response time for the k first results, (iii) number of contacted super-peers, (iv) volume of transferred data, and (v) number of transferred objects. Unless mentioned explicitly, we use the following default values: $d = 4$, $K = 50$, $10 \leq k \leq 50$, $n = 10^6$, $N_p = 2000$, while $N_{sp} = 10\% \times N_p$, and the dataset is uniform. We also assume 50KB/sec as network transfer bandwidth on connections between super-peers. The response time is measured as the sum of processing time and network transfer time required for the objects transferred in the network.

7.1 Performance of SPEERTO

We first examine the efficiency of SPEERTO on a network of $N_p = 2000$ peers, while we increase the dimensionality from $d = 2$ to $d = 10$. We set $K = 50$ and thus we can answer exactly any top-k query with $k \leq 50$. We obtain similar results for $N_p = 6000$, omitted due to space constraints.

The cost of skyline exchange is considered for uniform and clustered data. In Figure 6, the volume of transferred data per super-peer is depicted for varying dimensionality d. For the highest value $d = 10$, each super-peer induces on average 147MB of data in the network for the uniform dataset. The clustered dataset induces a slightly higher volume.

---

[3]Our implementation uses the XXL library available at: http://www.xxl-library.de

In Figure 7, the performance of SPEERTO is studied, in terms of volume of transferred objects, overall response time, and response time for the first k objects. First we examine the performance on uniformly distributed data and increase the dimensionality from $d = 2$ to 10, and we study top-10 to top-50 queries, assuming $K = 50$. In Figure 7(a), the response time is presented for different values of k. The plot illustrates the total response time taking into account the network delay, which depends on the size of transmitted data. We depict the results measured for uniform dataset with varying dimensionality. As expected the response time increases with the dimensionality. The increasing time with d is due to the fact that a) the size of transferred objects increases, and b) the processing time is higher, since the K-skyband and the skyline size increases with d as well. The cardinality of the K-skyband and the skyline influences the processing time of our threshold algorithm, which is executed on the objects stored on the querying super-peer. Thereafter we focus on the progressive property of SPEERTO, shown in Figure 7(b). The chart shows the response time for the first 10 results, for varying dimensionality. Notice that the first results are returned to the user immediately.

Figure 7(c) shows the number of contacted super-peers during top-k query processing. The number of contacted super-peers increases slightly with the dimensionality. Mainly the number of contacted super-peers depends on the number of objects k that are retrieved. For example if $k = 50$ objects are retrieved, more than 40 out of 200 super-peers are contacted. Since the dataset is uniformly distributed among the peers and therefore also among the super-peers, we can not avoid to contact all super-peers that contribute to the query result set.

In the following charts we study the effectiveness of the proposed threshold-based algorithm. The next two charts (7(d) and 7(e)) show the gain of threshold usage in SPEERTO. Figure 7(d) shows the improvement factor in terms of number of transferred objects, when threshold is used. The gain is very high, for example for $d = 6$ and top-k=50, only 50 objects are transferred, while without the threshold 1097 additional objects would have been transferred, i.e., the improvement factor is 21.9. Even though the gain in terms of number of transferred objects seems not to increase with the dimensionality, the benefit in terms of volume is higher, since the volume of the objects increases as the dimensionality increases. Figure 7(e) shows the percentage of super-peers that actually prune some objects that would otherwise be returned to the querying super-peer. In general, 90% of the queried super-peers manage to discard some objects, instead of returning them to the querying super-peer, due to the threshold.

In the next series of experiments we examine the proposed method's scaling features regarding data cardinality. Figure 7(f) depicts the response time while varying the cardinality of the dataset from $n = 1M$ to $n = 2.5M$. The slightly increasing response time with cardinality n is mainly due to higher processing times caused by the increase of the K-skyband and the skyline size. The number of contacted super-peers and transferred data are not influenced by the cardinality of the dataset.

We also study the scalability of SPEERTO with respect to the network size (Figure 7(g)). The network size N_p does not affect the overall response time, as the number of queried

(a) Response time with d (b) Response time for first 10 objects   (c) Number of contacted SPs

(d) Gain in number of transferred objects   (e) Percentage of SPs with gain   (f) Scalability with cardinality

(g) Scalability with N_p (h) Scalability with N_p (i) Clustered dataset
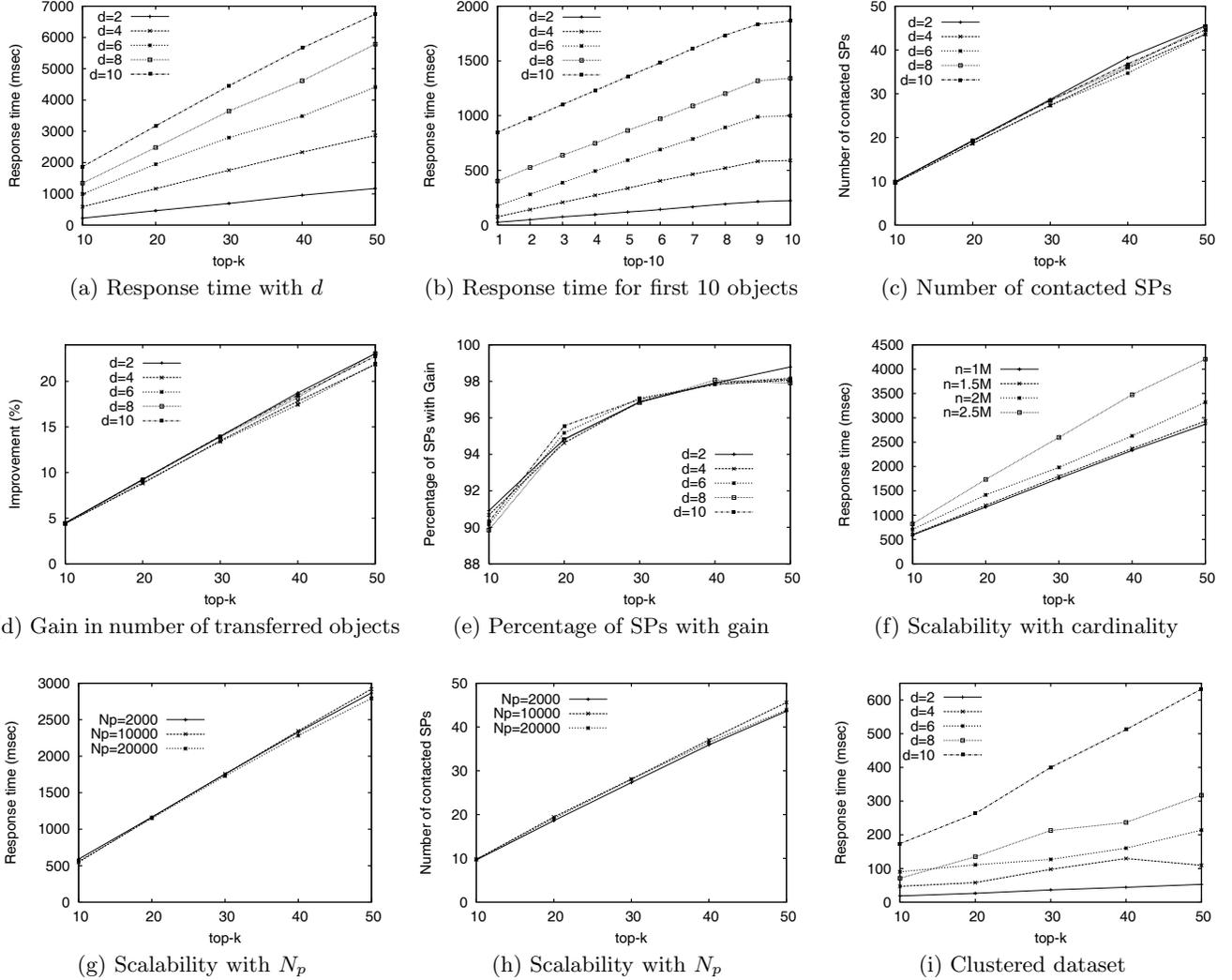
Figure 7: Performance of SPEERTO ($N_{sp} = 200$, $N_p = 2000$, $K = 50$)

super-peers (Figure 7(h)) remains practically the same for a given top-k query. In the next experiment, we study the performance on clustered data. Figure 7(i) depicts the response time for clustered data distribution while varying the dimensionality of the dataset. The clustered dataset leads to much smaller response time than the uniform distribution, because only a few super-peers are contacted during query processing. Therefore, SPEERTO performs better for the clustered dataset in terms of response time, contacted super-peers, and transferred data objects.

7.2   Top-k Queries with $k > K$

In the next experiments we evaluate the effectiveness of SPEERTO for top-k queries with $k > K$. We compare the actual top-k results with the ones retrieved from the K-skyband, by measuring the relative recall, i.e., the fraction of the produced top-k results that are in the true top-k results.

In Figure 8(a), we show the recall achieved for uniform distribution and cardinality 0.5M, while varying d from 2 to 4. The K parameter of the skyband is set to 10, which ensures us the exact results at least for any top-10 query. We

evaluate the performance of our approach for $10 \leq k \leq 200$. As expected recall decreases as k increases, but notice that the errors in the top-k list occur in the lower positioned objects, which are less important to the user. The skyband size is manageable and less than 1% of the dataset in any case. For $d = 2$ the skyband contains less than 100 points, thus recall decreases rapidly. For $d = 4$ the skyband size grows to less than 3500 data objects, and recall decreases less rapidly. Figure 8(a) shows that using the 10-skyband we can answer top-100 queries with recall around 40% for $d = 2$, while for $d = 4$ we can answer queries for higher k values with better recall. Experiments on clustered datasets resulted in similar recall values (Figure 8(b)). By varying the cardinality we noticed that recall (Figure 8(c)) and the skyband size (Figure 8(d)) are hardly influenced, which makes our approach feasible for large-scale systems.

In Figure 8(e) we vary K between 1 and 100 and evaluate the recall on top-100 queries for different dimensionality values. For $K = 1$ (which is the skyline set) we achieve recall of more than 90% for dimensionality 6 and 8, while the size

(a) Recall for uniform data   (b) Recall for clustered data   (c) Recall for varying n

(d) Size for varying n (e) Recall for varying K (f) Size for varying K
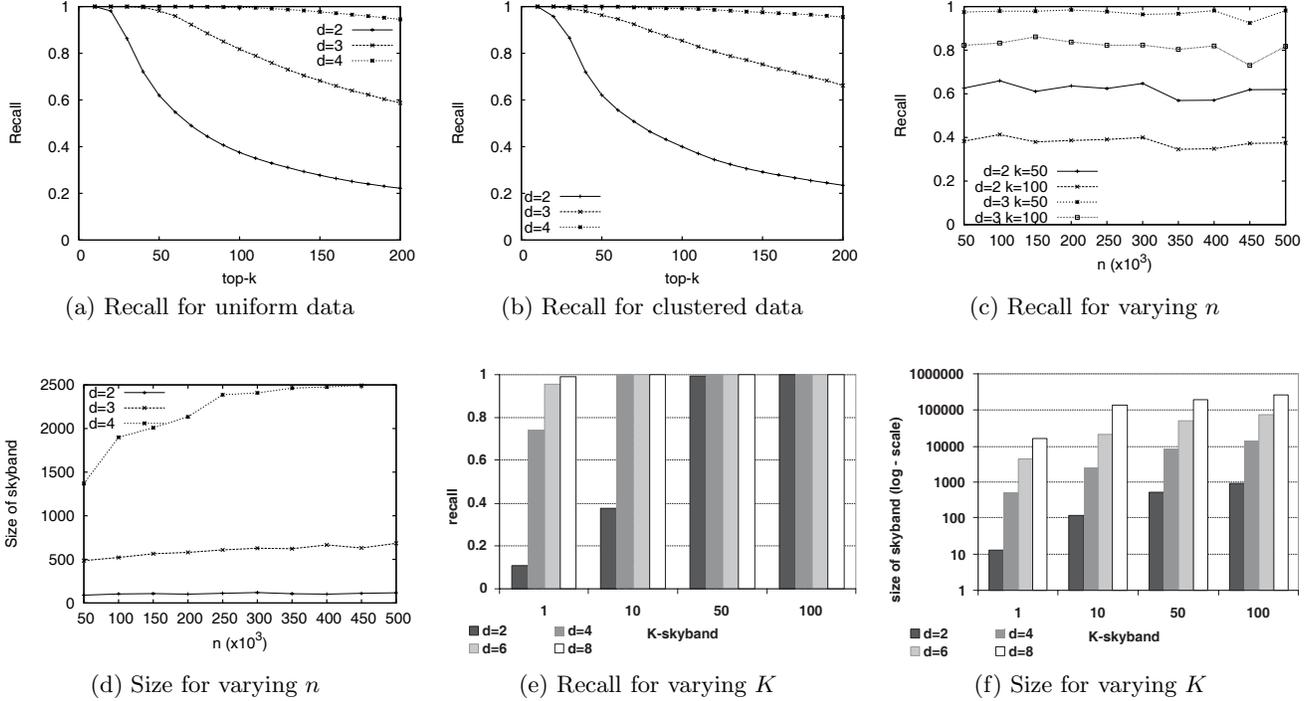
Figure 8: Top-k queries with $k > K$
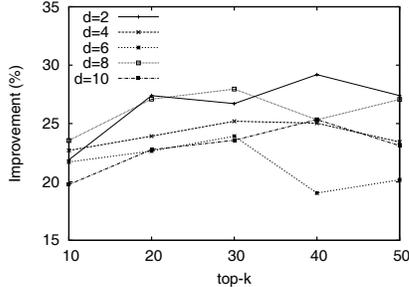
Figure 9: Parallel extension

of the skyline (Figure 8(f)) is up to 3% of the dataset size for $d = 8$. For $d = 2$ recall is low since the skyline consists of only 13 points. Recall values increase with dimensionality because the size of the K-skyband also increases.

7.3 SPEERTO Extensions

We also evaluate the merits of the SPEERTO extensions proposed. In Figure 9, we provide a comparative chart of the parallel version (PAR) of our algorithm compared to the original one. The results show a reduction in response time when the parallel variant is employed. The chart depicts the improvement percentage of our parallel approach for uniform datasets of different dimensionality. In average, the gain in response time is more than 20%, while the number of transferred objects is only marginally increased.

Finally, we evaluate the performance of the *abstract* skyline variant. In Figure 10(a), we depict the increase of the number of transferred objects when the abstract skyline vari-

ant is used. In this experiment we use uniform datasets of different dimensionality. The number of transferred objects increases rapidly, since the threshold is not used. The abstract skyline variant transfers up to 30% more data objects. In Figures 10(b) and 10(c), the gain of the abstraction in the case of data additions is depicted, for data dimensionality $d = 2$ and $d = 3$. The x-axis depicts the percentage of points added to the dataset, while the y-axis depicts the number of super-peers that have to update their skyline set. For example when $d = 2$ and 2% of the dataset is added, by keeping the 50% of the skyline points, only 4% of the super-peers need to update their skyline. In Figure 10(c), we notice that for $d = 3$ the number of modified super-peers increases, but again the gain of abstraction is significant.

8. CONCLUSIONS

In this paper we presented SPEERTO, a novel approach for answering top-k queries in a P2P network. Relying on a super-peer architecture, we proposed a threshold-based algorithm which forwards the top-k query requests among super-peers, in such a way that the amount of transferred data is minimized. For a maximum value of K, SPEERTO returns the correct answers for any top-k query ($k \leq K$), while supporting a large class of scoring functions. We proposed a variant of SPEERTO that queries in parallel more than one super-peers and an extension that restricts the cardinality of the skyline. We provided an extensive experimental evaluation showing that SPEERTO performs efficiently and provides a viable solution when a large degree of distribution is required. In addition, we studied experimentally the recall of top-k queries with $k > K$ and showed that we provide almost accurate results even for small K values.

(a) Variant with abstract skyline   (b) Data updates for $d = 2$ (c) Data updates for $d = 3$
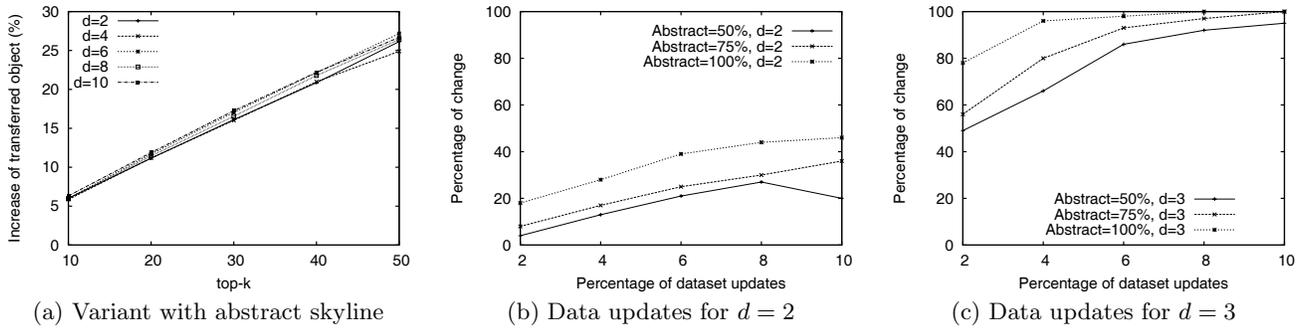
Figure 10: Abstract skyline extension of SPEERTO

9. REFERENCES

[1] R. Akbarinia, E. Pacitti, and P. Valduriez. Reducing network traffic in unstructured P2P systems using top-k queries. *Distributed and Parallel Databases*, 19(2-3):67–86, 2006.

[2] R. Akbarinia, E. Pacitti, and P. Valduriez. Best position algorithms for top-k queries. In *Proceedings of Int. Conf. on Very Large Data Bases (VLDB)*, pages 495–506, 2007.

[3] W.-T. Balke and U. Güntzer. Multi-objective query processing for database systems. In *Proceedings of Int. Conf. on Very Large Data Bases (VLDB)*, pages 936–947, 2004.

[4] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive distributed top-k retrieval in peer-to-peer networks. In *Proceedings of IEEE Int. Conf. on Data Engineering (ICDE)*, pages 174–185, 2005.

[5] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of IEEE Int. Conf. on Data Engineering (ICDE)*, pages 421–430, 2001.

[6] P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. In *Proceedings of Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 206–215, 2004.

[7] S. Chaudhuri, N. N. Dalvi, and R. Kaushik. Robust cardinality and cost estimation for skyline operator. In *Proceedings of IEEE Int. Conf. on Data Engineering (ICDE)*, page 64, 2006.

[8] S. Chaudhuri and L. Gravano. Evaluating top-k selection queries. In *Proceedings of Int. Conf. on Very Large Data Bases (VLDB)*, pages 397–410, 1999.

[9] S. Chaudhuri, L. Gravano, and A. Marian. Optimizing top-k selection queries over multimedia repositories. *IEEE Transactions on Knowledge and Data Engineering*, 16(8):992–1009, 2004.

[10] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *Proceedings of Int. Conf. on Data Engineering (ICDE)*, pages 717–719, 2003.

[11] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of Symposium on Principles of Database Systems (PODS)*, pages 102–113, 2001.

[12] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In *Proceedings of Int. Conf. on Very Large Data Bases (VLDB)*, pages 419–428, 2000.

[13] K. Hose, M. Karnstedt, K.-U. Sattler, and D. Zinn. Processing top-N queries in P2P-based web integration systems with probabilistic guarantees. In *Proceedings of International Workshop on Web and Databases (WebDB)*, pages 109–114, 2005.

[14] V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A system for the efficient execution of multi-parametric ranked queries. In *Proceedings of ACM Int. Conf. on Management of Data (SIGMOD)*, pages 259–270, 2001.

[15] J. Lu and J. Callan. Merging retrieval results in hierarchical peer-to-peer networks. In *Proceedings of the ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 472–473, 2004.

[16] J. Lu and J. Callan. Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In *Proceedings of European Conference on IR Research (ECIR)*, pages 52–66, 2005.

[17] A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Transactions on Database Systems*, 29(2):319–362, 2004.

[18] S. Michel, P. Triantafillou, and G. Weikum. KLEE: a framework for distributed top-k query algorithms. In *Proceedings of Int. Conf. on Very Large Data Bases (VLDB)*, pages 637–648, 2005.

[19] K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In *Proceedings of ACM Int. Conf. on Management of Data (SIGMOD)*, pages 635–646, 2006.

[20] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, 30(1):41–82, 2005.

[21] P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas, and D. Srivastava. Ranked join indices. In *Proceedings of IEEE Int. Conf. on Data Engineering (ICDE)*, pages 277–288, 2003.

[22] K. Zhao, Y. Tao, and S. Zhou. Efficient top-k processing in large-scaled distributed environments. *Data and Knowledge Engineering*, 63(2):315–335, 2007.