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

Distributed Top-k Query Processing by Exploiting Skyline Summaries

Akrivi Vlachou · Christos Doulkeridis · Kjetil Nørvåg

Received: date / Accepted: date

Abstract Recently, a trend has been observed towards supporting rank-aware query operators, such as top-k, that enable users to retrieve only a limited set of the most interesting data objects. As data nowadays is commonly stored distributed over multiple servers, a challenging problem is to support rank-aware queries in distributed environments. In this paper, we propose a novel approach, called DiTo, for efficient top-k processing over multiple servers, where each server stores autonomously a fraction of the data. Towards this goal, we exploit the inherent relationship of top-k and skyline objects, and we employ the skyline objects of servers as a data summarization mechanism for efficiently identifying the servers that store top-k results. Relying on a thresholding scheme, DiTo retrieves the top-k result set progressively, while the number of queried servers and transferred data is minimized. Furthermore, we extend DiTo to support data summarizations of bounded size, thus restricting the cost of summary distribution and maintenance. To this end, we study the challenging problem of finding an abstraction of the skyline set of fixed size that influences the performance of DiTo only slightly. Our experimental evaluation shows that DiTo performs efficiently and provides a viable solution when a high degree of distribution is required.

Keywords Top-k queries · skyline operator · distributed databases

1 Introduction

Due to the advent of large-scale data centers and cloud computing infrastructures, data generation and storage is becoming increasingly distributed. Thus, an emerging challenge is to support efficient query processing over data stored

Akrivi Vlachou · Christos Doulkeridis · Kjetil Nørvåg
Dept.of Computer Science, NTNU, Trondheim, Norway
E-mail: {vlachou,cdoulk,Kjetil.Norvag}@idi.ntnu.no

in a distributed network of collaborative computers. Even more important is to incorporate and support flexible query operators, such as top-k, that help 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 the best k objects only [10,19]. Numerous applications can significantly benefit from supporting top-k query processing, for example multimedia retrieval (including images) [11,17], digital libraries [22,23], web search [25], and e-commerce [24]. Consider for example online booking systems, e.g., for travel and accommodation, where the user is only interested in the best offers (air-tickets, hotels), while the relevant data is provided by multiple travel agencies, forming a distributed network.

In this paper, we focus on efficient top-k query processing in a generic distributed system. We present DiTo (Distributed Top-k Query Processing) a framework that supports top-k query processing over horizontally partitioned data stored on different servers. Our approach uses the skyline set [5] as a summary of the locally stored data on a server to effectively select and query the servers that store relevant data to a given top-k query. DiTo supports a large class of scoring functions that allow each user to define her own arbitrary preferences for each query. DiTo utilizes a thresholding scheme that enables selecting only relevant servers and avoiding transferring data that cannot belong to the result set. We show that our approach always provides the exact and complete result set in a progressive way, while only servers that may contribute to the top-k result are queried in a deliberate way. Finally, we extend DiTo in order to support data summaries of bounded size. To this end, we define an abstraction of the skyline set that consists of a fixed and user-defined number of points, but influences the performance of DiTo only slightly. To summarize, the main contributions of our work are:

- We study the applicability of the skyline operator used as data summarization for efficiently answering top-k queries, defined by a wide class of scoring functions, in generic distributed systems.
- We present DiTo, a framework for efficient distributed top-k processing that employs a thresholding scheme in order to facilitate pruning of objects and servers that cannot belong to the result set. We show that DiTo always retrieves the correct result set, while the number of transferred data and queried servers are minimized.
- We extend DiTo to support data summarizations of bounded cardinality. To this end, we define an appropriate set called *abstract skyline*, and study its properties thoroughly. Finally, we define the optimal abstract skyline problem, which leads to a skyline abstraction of high quality and preserves the efficiency of DiTo during query processing.
- Our experimental evaluation shows that DiTo provides a viable solution for distributed top-k processing, while also consistently outperforms a competitor algorithm.

The rest of this paper is organized as follows: In Section 2, the related work is presented. Section 3 provides the necessary definitions and preliminaries

for presenting our framework, while in Section 4, we describe our distributed model and provide an overview of our approach. Thereafter, in Section 5, our threshold-based top-k algorithm is presented. In Section 6, we study the abstraction of the skyline set and extend DiTo for supporting data summarizations of bounded size. Then, Section 7 describes maintenance issues related to our approach. Section 8 presents our experimental evaluation, and finally we conclude in Section 9.

2 Related Work

Approaches for distributed top-k query processing can be classified in two main categories. In the first category, the proposed approaches assume that the data is partitioned horizontally to the servers, which means that each server stores a fraction of the available data but all attribute values. In the second category, the related approaches assume vertically partitioned data and each server stores only some attributes of all available data. In the following, we first provide an overview of related work in centralized environments, and then review the distributed approaches of the two categories separately.

2.1 Top-k Queries in Centralized Databases

Several papers have dealt with the issue of top-k query processing in centralized database management systems [8,10,19,6,12,20,21]. Onion [8] precomputes and stores the convex hulls of data points in layers. Then, the evaluation of a linear top-k query is accomplished by processing the layers inwards, starting from the outmost hull. Prefer [19] uses materialized views of top-k result sets, according to arbitrary scoring functions. During query processing, Prefer selects the materialized view corresponding to the function that is most similar to the query scoring function, and examines a subset of the data elements in this view. In connection to top-k queries, the skyline operator [5] has received considerable attention. In [37] a layer-based indexing structure that relies on the dominance relationships, called Dominant Graph, was proposed to improve the performance of top-k query processing. In [29] the authors improve the performance of ranked join indices based on the concept of dominating sets. In [26] 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. Recently, reverse top-k queries [30,31] have been introduced, as a query that identifies the scoring functions (i.e., weighting vectors) that make a point a top-k result. Extensions of reverse top-k queries have also been proposed for mobile environments [32] and for identifying influential products [33].

2.2 Distributed Top-k Queries over Horizontally Partitioned Data

As the main topic of this paper is distributed top-k query processing in the case of horizontal data distribution, these papers are the most relevant to our work. Despite the wide applicability of top-k queries over horizontally partitioned data, only few works have addressed this problem 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 servers, 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 by Akbarinia *et al.* [1], where the main technique is a variant of flooding, followed by a merging score-list step at intermediate peers. Zhao *et al.* [36] rely on result caching to prune network paths and answer queries without contacting all peers. Even though a new top-k query can sometimes be answered from the cache, the performance of their approach depends on the query distribution due to the use of caching. Hose *et al.* [18] 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 are required. In [14], an approach is proposed that tries to minimize the users' waiting time of top-k results, at the expense of multiple phases of data transmission. Recently, Ryeng *et al.* [27] studied caching of top-k results and the use of remainder queries to answer future top-k queries.

The applicability of the skyline operator for efficiently routing top-k queries over a super-network was studied in [35,34]. This paper differs to the earlier work by Vlachou *et al.* [35,34], because it assumes a generic distributed architecture and not a super-peer network. Furthermore, in this paper we study in details the problem of restricting the cardinality of the data summarization, which was only briefly tackled in the earlier work. Finally, we provide a detailed experimental evaluation verifying the applicability of the proposed approach for generic distributed architectures.

In the relevant area of distributed 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 the quality of document retrieval (rather than on performance) and on defining an appropriate scoring function. For example, in [22,23], 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.

Symbols	Description
d	Data dimensionality
n	Data set cardinality
p, q	Data points
$p[i]$	Value of data point p in i-th dimension d_i
N	Number of servers
S_i	The i-th server
O_i	Data set of the i-th server S_i
SKY_i	Skyline set of the i-th server S_i
f	Scoring (top-k) function
k	Value of k
w	Weighting vector that defines a top-k query
$q_k(f)$ or $q_k(w)$	A top-k query
$\widehat{SKY}$	Abstract skyline set
B	Size of abstract skyline $\widehat{SKY}$

Table 1 Overview of symbols

2.3 Distributed Top-k Queries over Vertically Partitioned Data

Previous work in distributed top-k query processing [11,17,24] has focused on vertically distributed data over multiple sources, where each source provides a ranking over some attributes. Most approaches, such as [2], try to improve some limitations of the Threshold Algorithm [15]. 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.* [24] study top-k query evaluation over web-accessible databases, including random accesses to score lists, instead of sorted accesses only, as in [15]. In [7], 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 [25]. 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.

3 Preliminaries

In this section, we present the data model, the definition of top-k queries and basic related concepts, as well as some important properties of top-k queries. An overview of the most important symbols used can be found in Table 1.

3.1 Data Model

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

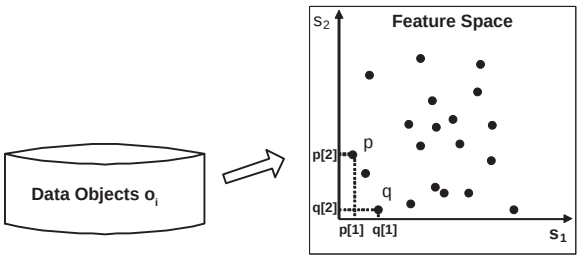


Fig. 1 Feature space

s_j are numerical scoring functions with non-negative values that evaluate certain attributes of database objects. For example, s_j can be extracted features, aggregations of attribute values, 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, hence it is modeled as 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. Fig. 1 depicts a 2-dimensional example. In the rest of this paper, we use the terms object and data point interchangeably.

3.2 Top-k queries

Top-k queries are defined based on a scoring function f that aggregates the individual scores of different dimensions into an overall scoring value, that in turn enables the ranking (ordering) of the data points. The result of a top-k query is the ranked list of the k data objects with lowest *score* values.

Definition 1 *(Top-k query)* Given a positive integer k and a user-defined scoring function f, the result set TOP_k of the top-k query is a set of points such that $TOP_k \subseteq O$, $|TOP_k| = k$ and $\forall p_1, p_2 : p_1 \in TOP_k$, $p_2 \in O - TOP_k$ it holds that $f(p_1) \leq f(p_2)$.

We assume that the scoring function f is increasingly monotone, i.e., if $p[i] \leq p'[i]$ for every i, then $f(p) \leq f(p')$. The restriction of monotonicity is a common property [10,15] 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 most important and commonly used case of monotone scoring functions is the weighted sum function, also called linear. Each dimension d_j has an associated query-dependent weight w_j indicating d_j's relative importance for the query. The aggregated score for point p is defined as a weighted sum of the individual scores: $score(p) = \sum_{j=1}^{d} w_j \cdot p[j]$, 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, top-k queries with respect to only to a subset of the available features can be defined. The

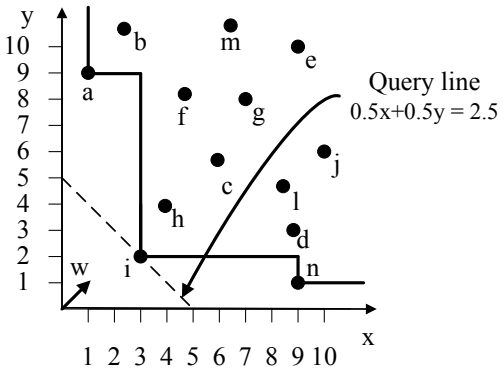


Fig. 2 Skyline and top-k queries

weights represent the relative importance between different dimensions, and without loss of generality[1] we assume that $\sum_{i=1}^{d} w[i] = 1$. The weights indicate the user preferences and they determine the ordering of the data objects and therefore the top-k result set.

Consider for example the data set depicted in Fig. 1. Assigning a high weight to feature s_2, means that the value of feature s_2 is more important for the user, and therefore point p is the top-1 object, which has the minimum s_2 value. On the other hand, if a low weight is used, point q becomes the top-1 object.

Even though our approach is applicable for any increasingly monotone aggregate function, we use the weighted sum function in our examples. In general, 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 query can be denoted as $q_k(w)$, where w is a d-dimensional vector $w = \{w_1, \ldots, w_d\}$. Notice that both the scoring function f 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 f.

Consider a linear top-k query defined by a vector w. In a d-dimensional Euclidean space, there exists a one-to-one correspondence between a weighting vector w and a hyperplane which is perpendicular to w. We call the (d-1)-dimensional hyperplane, which is perpendicular to vector w and contains a point p as the *query plane of w crossing p*. All points lying on the query plane, have the same scoring value equal to the score $f(p)$ of point p. Fig. 2 depicts an example, where the query plane (equivalent to a *query line* in 2d) is perpendicular to the weighting vector $w = [0.5, 0.5]$. All points lying on the query line have a score value $f(p_i) = 2.5$. Furthermore, the rank of a point p based on a weighting vector w is equal to the number of the points enclosed in the half-space defined by the query plane that contains the origin of the data

[1] As discussed in [29] the magnitude of the query vector does not influence the query result as long as the direction remains the same.

space. If we consider the 2-dimensional example depicted in Fig. 2, the point i is the top-1 object for the query $0.5 \cdot x + 0.5 \cdot y$.

3.3 Relation Between Top-k and Skyline Queries

In the following, we define the skyline set and discuss its relation to top-k queries.

Definition 2 *(Skyline query)* 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 Fig. 2, a, i and n are the skyline points. The following auxiliary lemma [34] reveals the way dominance relationships affect the relative ranking between two data points.

Lemma 1 *For any increasingly monotone function it holds that if p dominates q, then $f(p) \leq f(q)$.*

The following lemma [34] is essential in order to exploit the skyline set for facilitating top-k computation.

Lemma 2 *[34] The top-1 object for any increasingly monotone function belongs to the skyline set.*

In Fig. 2, notice that i, which is the best match for the top-1 query with $w = [0.5, 0.5]$, belongs to the skyline.

4 System Overview

The overall aim is to exploit the intrinsic relationship between top-k and skyline queries in order to facilitate efficient top-k query processing in generic distributed setting. In the following, we first describe the system model in detail, and then we describe basic approaches for query processing.

4.1 System Model

We assume that a set of N servers S_i participate in the system, and each server S_i stores locally a set of n_i d-dimensional points, denoted as a set O_i ($1 \leq i \leq N$). Since we assume horizontal data distribution, the size of the complete set of points is $n = \sum_{i=1}^{N} n_i$ and the data set O is the union of all sets of points O_i stored locally at any server S_i ($O = \bigcup O_i$).

In addition, there exists a *coordinator* server S_C that coordinates the query processing. A top-k query can be initiated by any server, and the query request is forwarded to the coordinator S_C that is responsible for query execution by propagating the top-k query to those servers that store relevant data and gathering the final result set. Then, the result set is send back to the server that posed the query.

In general, each server is able to open direct connection to the coordinator server S_C, using its IP address. However, notice that the approach is also applicable, when no direct connection between servers can be established, and the communication is achieved by query forwarding through other servers (for example in the case of servers connected in a peer-to-peer topology).

4.2 Basic Approaches for Query Processing

A first naive approach is that S_C sends the query to all servers in order to retrieve the individual top-k results and combine them to produce the top-k result of the query. The advantage of this approach is that no knowledge about the data stored at the different servers is required at S_C a priori. However, even though contacting all servers is appropriate for gathering metadata needed for query processing, it is too costly to employ for each individual query at query time. Hence, this approach does not scale with the number of participating serves, which is expected to be significantly larger than k, thus leading to much more data being transferred than the requested k data objects.

An alternative naive approach is that each server broadcasts its K-skyband[2] to the coordinator before query processing in a proactive way. Then, the coordinator has enough data to answer any top-k query with $k \leq K$ locally. The advantage of this approach is that the query can be processed without contacting any remote servers. However, this approach is not feasible in a highly distributed environment, because of the non-negligible size of the skyband compared to the size of the data set, as well as due to the significant cost of maintenance in the presence of data updates.

4.3 Overview of DiTo

In order to combine the advantages of the aforementioned approaches while alleviating their disadvantages, we propose an efficient approach, called *DiTo* (`Distributed Top-k`), that supports top-k queries over distributed data by exploiting the intrinsic relationship between top-k and skyline queries. The main challenge addressed by DiTo is processing top-k queries over the data stored on different servers, in a way that only servers that may contribute to the query

[2] The K-skyband is the set of points which are dominated by at most $K-1$ other ones. 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.

are contacted. Our technique guarantees accurate results, while minimizing the number of queried servers and the amount of network traffic.

As already mentioned, each server S_i maintains its own data objects locally. In DiTo, servers publish only few selected data points, namely the skyline points, to the coordinator S_C as a data summarization, while the original data is stored at the server S_i. Intuitively, the skyline is the border with respect to the axes of the data stored locally. DiTo uses the skyline set in order to bound the score of the locally stored data and employs a threshold-based distributed algorithm (see Section 5) to efficiently compute the top-k result. In the more general case, DiTo is applicable by publishing only an abstraction the skyline set, called *abstract skyline*, that contains a limited number of points per server.

Notice that the cardinality of the skyline is significantly smaller than the cardinality of the K-skyband, while the abstract skyline has a user-defined size. It should be stressed that even though the skyline operator is CPU-intensive [9] and therefore more costly than a top-k query, DiTo 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 and scoring function f can be processed.

5 DiTo: Distributed Top-k Query Processing

In this section, we first introduce a threshold-based algorithm for distributed top-k query processing that queries carefully selected servers to produce the correct top-k result set. Then, we also show that our threshold-based algorithm is optimal in terms of the number of contacted servers and the volume of transferred data.

5.1 Threshold-based Top-k Algorithm

Our distributed top-k algorithm assumes that a server that joins the distributed system first computes the skyline set of its locally stored data, and then sends its skyline set to the coordinator server S_C. Thus, the coordinator server S_C assembles N sets of skyline points SKY_i, $(1 \leq i \leq N)$. These points are called *summary objects*. In the following, we describe our threshold-based top-k algorithm assuming that S_C stores the summary objects SKY_i of all servers.

Description: Let us first consider a two-dimensional space as a showcase scenario as illustrated in Fig. 3, which depicts the information that is available to the coordinator server S_C, when query processing starts. In more details, the skyline sets of three servers S_1, S_2 and S_3 are shown, which are the summary objects stored at the coordinator server S_C. As described in more detail below, the actual data objects will be transferred to the coordinator server S_C from the corresponding servers during query processing.

In a two-dimensional space, given an arbitrary weighting function that defines the slope of a query line, progressive processing of the top-k query is

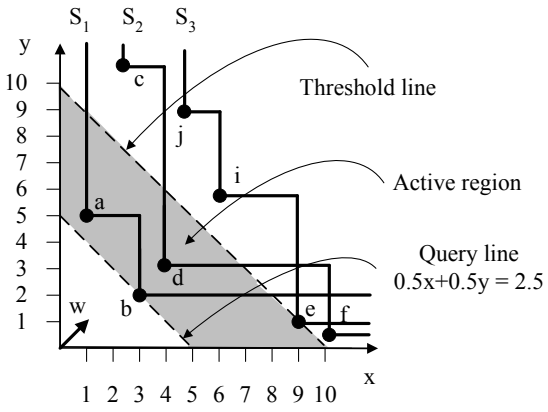


Fig. 3 Example of query line

similar to sweeping the query line through space from the origin of the axes towards the data. The first data object that the line meets is the top-1, the second the top-2, and so forth, until top-k data objects are retrieved. Each time the line meets a data object, this object can be immediately returned to the user, as it is definitely the next top result of the query (progressive property of our algorithm). Instead, when the line meets a summary object, the owner server S_i needs to be contacted and those of its data objects that are necessary for processing are transferred from S_i to S_C, thus replacing S_i's summary objects. At each step, the query line indicates how far the data space is examined, which means that there does not exist any other object in the examined space that has not been retrieved yet. This guarantees that there is no data object that has a better (smaller) scoring value than the retrieved objects.

A threshold value is defined as the score of the k-th summary or data object encountered so far. In the two-dimensional space, this score defines a *threshold line*, of identical slope with the query line, which gradually sweeps the space towards the origin of the axes. The region defined between the query and the threshold line 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 S_C is not aware of all data points enclosed in the active region, therefore if a summary object is retrieved, the corresponding server must be queried.

Continuing the example depicted in Fig. 3, consider a linear top-4 query with weights $w = (0.5, 0.5)$. The top-4 objects (b, a, d and e) based on the summary objects stored on S_C are retrieved, and the score of the 4th object (e), defines the threshold line. This guarantees that the results of this top-k query are found in the active region. Notice that some data points of S_1 or S_2 may fall in the active region and therefore point e may not belong to the top-4 result set. First, the summary object b is examined and since it belongs

Algorithm 1 Query processing on S_C

```
1: Input: Query q_k(f)
2: list = S_C.query_{∪SKY_i}(q_k(f))
3: threshold = f(list[k])
4: c = 0
5: while (c < k) do
6:    next_obj = list.pop()
7:    if next_obj is a summary object then
8:       S_i = next_obj.server()
9:       temp = S_i.query(q_{k-c}(f), threshold)
10:      list.removeSummaryObj(S_i)
11:      list.add(temp)
12:   else
13:      return next_object to the user
14:      c = c + 1
15:   end if
16:   threshold = f(list[k - c])
17: end while
```

to S_1, the query is forwarded to this server and some of the data points stored at S_1 are send back to S_C. In the next step, the data object b is retrieved and returned to the user as top-1 result. Then, data point a is retrieved and returned as the top-2 result. Afterwards, we retrieve the summary object d that belongs to S_2. Therefore, server S_2 is queried and d is returned to the user. Assuming that another data point of S_2 is enclosed in the active region, that point will be returned to the user and the query processing stops. Otherwise, if no other point is enclosed in the active region, S_3 is queried and point e is returned to the user.

Algorithmic description: Algorithm 1 describes our method (DiTo) for distributed top-k query processing. The summary and data objects retrieved so far are kept in a sorted list based on the scoring value. In the case of score ties, data objects precede summary objects in the list. This list is initialized (line 2) by the coordinator server (S_C) with the top-k objects of the skyline sets $\cup SKY_i$ of all servers.

Then, a threshold is set based on the scoring value of the k-th object (line 3), since any object with higher score cannot belong to the final result set. In each iteration, the top object (line 6) of the list is examined (lines 5–17). If the top object is a summary object (lines 7–12), then the server (S_i) responsible for this summary object is queried (line 8). A subtle situation occurs in the case of score ties of summary objects belonging to different servers. In this special case, the server that will be contacted is the one with higher number of summary objects with that particular score.

In more details, a top-$(k - c)$ query is posed to S_i along with the current threshold value (c denotes the number of results returned thus far to the user). Then S_i sends back $k - c$ data objects, or fewer if there do not exist $k - c$ objects with better (smaller or equal) value than the threshold. After S_i's data objects are retrieved by S_C, all summary objects of S_i are removed from the

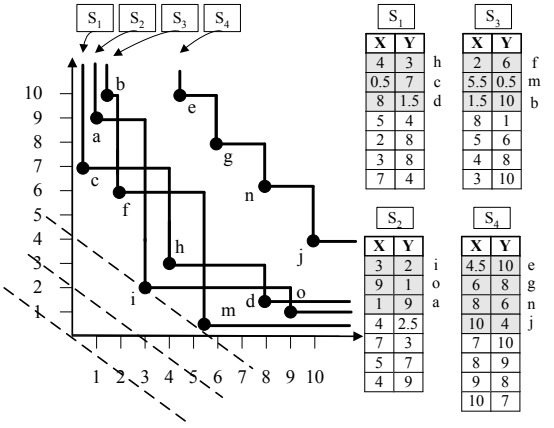


Fig. 4 Example of distributed algorithm for top-k query processing for 4 servers $S_1, \ldots, S_4$.

sorted list (line 9) before inserting its data objects (line 10), since they need not be maintained any longer.

In each subsequent iteration, if a data object is retrieved (12-15), it is returned to the user as the top-1, top-2, etc. result. Finally, the threshold is updated (line 16) with the scoring value of the $(k - c)$-th object in the list. The algorithm terminates when k data objects have been retrieved from the sorted list.

Example 1 Consider a small distributed system of four servers S_1–S_4, and the coordinator server S_C that has assembled the skyline sets of all servers, as depicted in Fig. 4. Let us assume that S_C 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 data points stored on each server are depicted in tables. The grey-shadowed objects are the skyline objects that are broadcast to S_C and they are also depicted on the left part of the figure graphically. According to Algorithm 1, the sorted list is initialized with summary objects: i(3,2), m(5.5,0.5) 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 server S_2. Thus S_C sends a top-3 query to S_2, and retrieves its local (at S_2) 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 S_C since they are discarded by the threshold value. So, only two points are returned to S_C by S_2 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 the list becomes of size 2 and it contains: m(5.5,0.5), (4,2.5). Next m is processed and as it belongs to S_3, S_C sends a message to S_3 requesting its top-2 objects. S_3 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 Analysis of DiTo

DiTo progressively returns accurate and exact answers for any top-k query. Moreover, the usage of the threshold ensures that the communication costs is reduced by preventing unnecessary data objects from being transferred to S_C during query processing. DiTo also avoids querying servers that do not contribute to the result set. In the following, we assume that the query is answered using a snapshot of the system, i.e., fixed servers and static contents, and we show the correctness of our algorithm and the optimality in terms of queried servers and transferred data.

Lemma 3 *(Correctness of Algorithm 1) Let TOP_k be the set of points returned by DiTo as the top-k result set for a given scoring function f. Then, it holds that for any $q \in TOP_k$: $\nexists p \in O_i$ such that $p \notin TOP_k$ and $f(p) < f(q)$.*

Proof The proof is by contradiction. Let us assume that $\exists p \in O_i$ such that $p \notin TOP_k$ and $f(p) < f(q)$, where $q \in TOP_k$. We distinguish two cases.

- Server S_i is not queried. Then, there exist k data points p_j with better score than any of the skyline points of S_i. Thus, $f(p_j) \leq f(p_{sky})\ \forall p_{sky} \in SKY_i$, $1 \leq j \leq k$. Lemma 1 guarantees (for any increasingly monotone function) that for at least one $p_{sky} \in SKY_i$ it holds that $f(p_{sky}) \leq f(p)$. Thus, $f(p_j) \leq f(p_{sky}) \leq f(p)$, $1 \leq j \leq k$. Moreover, since $f(p) < f(q)$, then $f(p_j) < f(q)$ $(1 \leq j \leq k)$, which means that $q \notin TOP_k$. This leads us to a contradiction.
- Server S_i is queried. Since $p \notin TOP_k$, p is not retrieved from S_i as one of the $k - c$ data objects with best scores, i.e., p is not retrieved by S_i.query$(q_{k-c}(f), threshold)$ (line 1). Notice that in this case c $(c < k)$ data points have already been returned to the user as the top-c results. Again, we distinguish two cases:
  - If $f(p) \leq threshold$, and since p is not retrieved from S_i, then there exist $k - c$ data points $p_j \in O_i$ for which it holds that $f(p_j) \leq f(p) < f(q)$. This means that there exist k in total data points with better score than q, thus $q \notin TOP_k$. This leads us to a contradiction.
  - If $f(p) > threshold$, then there exist $k - c$ summary objects or data points in the list, such that $f(p_j) \leq f(p) < f(q)$. Together with the already returned c data points, this means that there exist k in total data points with better score than q, thus $q \notin TOP_k$. This leads us again to a contradiction.

Let $\mathcal{A}$ denote the the class of deterministic algorithms that retrieve correctly the result set of any distributed top-k query. Obviously, DiTo $\in \mathcal{A}$ Furthermore, we denote as $\mathcal{DT}$ the family of threshold-based algorithms that

contact each server at most once as described by Algorithm 1. Each algorithm of the family is parameterized by a different function for threshold and it holds that DiTo $\in \mathcal{DT}$. The definition of the family $\mathcal{DT}$ as the set of all instantiations of all threshold-based algorithms, will be used to show that the proposed threshold is tight. We consider two performance metrics and the associated notions of optimality. The first metric is the number of servers that are contacted in order to retrieve the correct result set. It is important for any algorithm to minimize the number of contacted servers to ensure scalability with the number of queries posed in the system. The second metric is the number of transferred data objects during query processing. This metric should also be minimized in order to minimize bandwidth consumption, which is an important concern, especially in bandwidth-constrained networks.

Theorem 1 *DiTo is optimal within $\mathcal{A}$ based on the number of contacted servers.*

Proof Let us denote as $S_1, \ldots, S_j$ the order in which DiTo contacts j servers in order to retrieve the correct result set. Let us assume that there exists an algorithm $A \in \mathcal{A}$ that contacts fewer than j servers and always retrieves the correct result set. Then, there exists at least one server $S_i \in \{S_1, \ldots, S_j\}$ that is not contacted by A. Since S_i was contacted by DiTo, there exists at least one $p_{sky} \in SKY_i \subseteq O_i$ stored at S_i, such that fewer than k data objects p with $f(p) \leq f(p_{sky})$ exist in $O_1 \bigcup \cdots \bigcup O_{i-1}$. Also, since the data and summary objects are accessed sorted based on $f()$ values, there does not exist any p in $O_{i+1} \bigcup \cdots \bigcup O_j$ such that $f(p) < f(p_{sky})$. Hence, $p_{sky} \in TOP_k$, and A fails to retrieve it, which is a contradiction because A is a correct algorihtm.

To complete the proof, we need to examine the case that there exist data objects p' in $O_{i+1} \bigcup \cdots \bigcup O_j$ such that $f(p') = f(p_{sky})$. Based on the definition of top-k query, in order to be a correct algorithm, A must report any of these objects p' instead of p_{sky}. However, to report any of these objects A needs to contact the corresponding server $S'_i \notin \{S_1, \ldots, S_i\}$, hence A will contact as many servers as DiTo, which is a contradiction.

Theorem 2 *DiTo is optimal within $\mathcal{DT}$ based on the number of transferred data objects.*

Proof Let us assume that there exists a correct algorithm $A \in \mathcal{DT}$ that employs a different threshold than DiTo and transfers fewer data objects. Let p be a data object that is transferred by DiTo and not by A. Since DiTo transfers p, the list has at most $k-1$ data or summary objects with better score than p. We will construct a data set that will lead to exactly the same list, but A will fail to return the correct result for this data set. Assume that any server S_i that corresponds to a summary object in the list stores only this object. Then, A will replace each summary object by the corresponding skyline point and fewer than k objects will have a better score than p. To be a correct algorithm, A must either report p (since $p \in TOP_k$), or report another point p' with $f(p') = f(p)$. The first case leads us to a contradiction, since A does not

transfer p thus A cannot report p as top-k result. In the second case, A cannot avoid transferring another point p' (instead of p), which contradicts with the assumption that A transfers fewer points than DiTo.

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 from being transferred during query processing, and c) the number of queried servers is minimized.

5.3 Parallel Variant of DiTo

DiTo minimizes the number of servers contacted and the amount of transferred data at the expense of accessing servers one at a time. Obviously, in certain cases, this may have a negative effect in total execution time (even though the response time will still be acceptable as the algorithm is progressive). For example, when the network transfer rate is small or when the latency in communication is non-negligible compared to the processing cost, then the performance of DiTo in terms of total execution time may degrade.

In order to address such cases and reduce the response time, we propose a variant of DiTo that queries in parallel more than one server each time. This variant of DiTo sacrifices the above notions of optimality, in order to improve the total execution time. The amount of transferred objects increases due to the fact that multiple servers will be queried using a common threshold value, which could have been progressively refined based on returned results, had the servers been contacted one at a time. The number of contacted servers increases because from those servers queried in parallel, some may not return top-k results.

The only remaining issue is to determine the number of servers that should be queried in parallel. Using a high value may lead to contacting servers in vain, whereas using a small value may result in multiple communication phases, thus affecting the total time. Various approaches can be used to tackle this issue. A simple solution is to define a constant number $g(k)$ of servers as a function of k. Another approach is to maintain statistics from previous top-k queries about the distribution of results, and use them to estimate the appropriate number of servers to query in parallel. Instead, we use an approach proposed in [34] that requires no statistics of previous queries. The essence of this approach is that first a single server is queried and based on the threshold and the scores of its top-k results, we estimate the number of servers that need to be contacted to produce the correct top-k result set. The interested reader can find the details of this approach in [34].

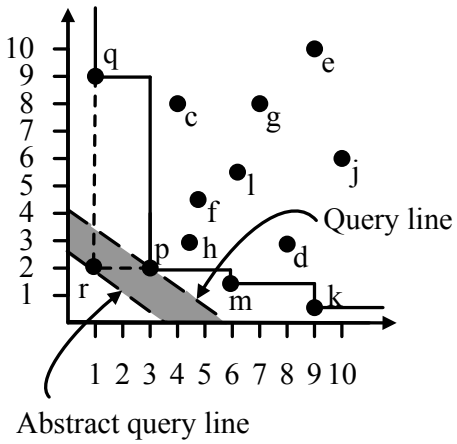


Fig. 5 Rationale of abstract skyline

6 Bounding the Cardinality of Summaries

In our framework, the skyline set is used as data summarization for top-k queries. However, it is well-known that the cardinality of the skyline set may be extremely high [9], often comparable to the size of the data set, especially for high-dimensional or anti-correlated data sets. In the case of DiTo, this would result in high storage requirements at S_C, but more importantly in high construction and maintenance cost. Moreover, local query processing at S_C becomes more expensive as the number of summary objects increases. To handle this shortcoming of the skyline operator, we extend DiTo to support data summaries that consist of an abstraction of the skyline set of fixed size. The aim is to bound the cardinality of the summaries, while maintaining an acceptable level of performance for DiTo.

In the following, we sketch the rationale of the abstraction (Section 6.1), we formally define the *abstract skyline* and discuss its theoretical properties as well as the quality of the skyline abstraction (Section 6.2), we present an algorithm that computes the abstract skyline set (Section 6.3), and finally we show how DiTo can be adapted to work with the abstract skyline set(Section 6.4).

6.1 Rationale and Example

The problem is to find an abstraction of the skyline set of fixed size to send to S_C that still allows S_C to select the servers that store relevant data to a given query. This abstraction needs to have two main properties. First, the skyline abstraction must ensure that the correctness of DiTo is not violated. Secondly, the skyline abstraction must enable effective server selection, i.e., avoid contacting servers that cannot contribute to the top-k result.

Consider for example the data set depicted in Fig. 5, where the points q, p, m, and k represent the skyline objects. Given a upper bound B=3 for the

skyline abstraction, let us assume that the skyline abstraction consists of the points r, m, and k, i.e., the points q, p are substituted by point r. DiTo selects the servers that are contacted during query processing based on the summary objects. To avoid violating the correctness of DiTo, the skyline abstraction must guarantee that whenever a server S_i is contacted based on the skyline points, this server S_i will be also contacted based on the skyline abstraction. Notice that in the example of Fig. 5, the query line meets point r before points p and q for any top-k query. Thus, the correctness of DiTo is not violated.

On the other hand, the skyline abstraction may lead to more contacted servers and more transferred data. Assume that the depicted skyline points in Fig. 5 represent the summary objects of a server S_i that are available at S_C. Then, if a skyline point of another server S_j is enclosed in the shadowed area, then S_j would have been first contacted based on the skyline points. In the case of the abstract skyline, S_i is contacted before S_j, which could have been avoided (e.g., in the case that S_j could deliver the k-th object of the top-k result set). Consequently, it is clear that the abstraction causes an increase in the number of contacted servers (and transferred data). Obviously, there is a trade-off between the abstraction size and the accuracy of the skyline abstraction. Intuitively, the larger the shadowed area in Fig. 5, the higher the probability of querying more servers. This shadowed area depends on the scoring function, however the quality of the abstract skyline should be defined independently of the scoring function.

6.2 Definitions and Theoretical Properties

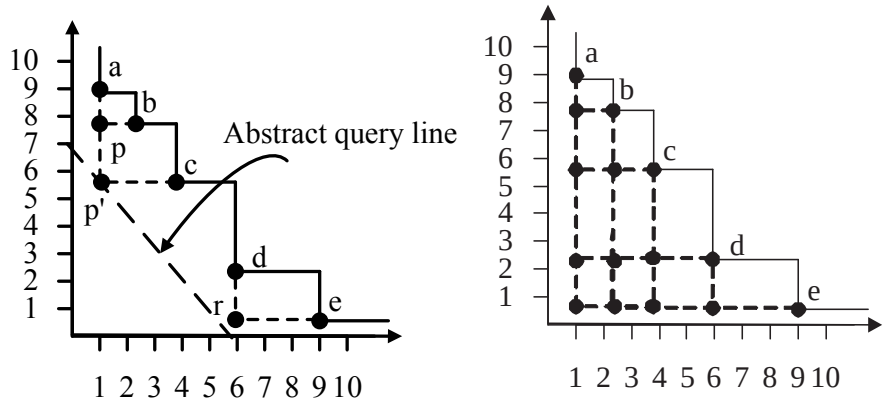
In the following, we define a suitable abstraction of the skyline set, called *abstract skyline*. Recall that m denotes the cardinality of the skyline set SKY, while B denotes the cardinality of the abstract skyline.

Definition 3 (*Abstract Skyline*) Given an upper limit B, we define the *abstract* skyline $\widehat{SKY}$ of a skyline set SKY, as a set of at most B points ($B \leq m$), such that: a) each point $p \in SKY$ is either dominated by or equal to at least one point $q \in \widehat{SKY}$ and b) $|\widehat{SKY}| \leq B \leq m$.

Notice that the points $q \in \widehat{SKY}$ are not necessary data points of the data set. The following lemma shows that the summary objects bound the scores of the locally stored data, when the abstract skyline is used as data summarization instead of the skyline set.

Lemma 4 *For any $p \in O_i$ it holds that $\exists q \in \widehat{SKY}_i$ such that $f(q) \leq f(p)$.*

Proof We distinguish two cases. If $p \in SKY_i$, then based on Definition 3 it is either dominated by or equal to at least one point $q \in \widehat{SKY}_i$. Due to Lemma 1, it holds that $f(q) \leq f(p)$. If $p \notin SKY_i$, then there exists a point $p' \in SKY_i$, such that p' dominates p. Due to Lemma 1, it holds that $f(p') \leq f(p)$, and based on Definition 3 there exists $q \in \widehat{SKY}_i$ that is either equal to or dominates p'. Thus, $f(q) \leq f(p') \leq f(p)$.



(a) Example of partial order of abstract skyline (b) Candidate abstract skyline points

Fig. 6 Examples of abstract skyline

Lemma 4 shows that the abstract skyline points can be used for selecting relevant to the query servers, since the scores of the locally stored data are bounded by the scores of the abstract skyline points. Intuitively, it means that when the query line meets a skyline point of a server S_i, it has always already met an abstract skyline point belonging to S_i before the skyline point. Obviously, there exist infinite abstract skyline sets by definition. To ensure that the efficiency of DiTo is influenced only slightly, we define a partial order between different abstract skyline sets that captures the quality of the abstract skyline set.

Definition 4 (*Partial Order of Abstract Skyline*) Given two abstract skyline sets $\widehat{SKY}$ and $\widehat{SKY}'$, if it holds that $\forall p \in \widehat{SKY}$, $\exists p' \in \widehat{SKY}'$ such that p' dominates p or $p' = p$, then $\widehat{SKY}$ is better than or equally good to $\widehat{SKY}'$ ($\widehat{SKY} \preceq \widehat{SKY}'$).

The intuition of the partial order is that the abstract skyline should bound the skyline points as tight as possible. For example, consider the example of Fig. 6(a). Assuming two valid abstract skyline sets with B=3, namely $\widehat{SKY} = \{p, c, r\}$ and $\widehat{SKY}' = \{p', c, r\}$, then it holds that $\widehat{SKY} \preceq \widehat{SKY}'$ based on the partial order. This is in accordance with the fact that any query line always first meets p' and then p, and therefore deteriorates the efficiency of DiTo.

In the following, we show that even though there exist infinite abstract skyline sets (by definition), the abstract skyline points that lead to a good data summarization are finite.

Theorem 3 *Given an abstract skyline $\widehat{SKY}$, if it does not hold that $\forall p \in \widehat{SKY}$ and $\forall d_i \in D$: $\exists p' \in SKY$ such that $p[i] = p'[i]$, then there exists another abstract skyline $\widehat{SKY}'$, such that $\widehat{SKY}' \preceq \widehat{SKY}$.*

Proof Let us assume that $\exists p \in \widehat{SKY}$, such that $\nexists p' \in SKY$ with $p[i] = p'[i]$, $d_i \in D$. Then, let $\{q_j\}$ be the set of points such that $q_j \in SKY$ and p dominates q_j. We define the abstract skyline set $\widehat{SKY'} = \{\widehat{SKY} - p\} \bigcup q$, where q is the point defined as $q[i] = min_{\forall q_j}(q_j[i])$, $\forall d_i \in D$. The set $\widehat{SKY'}$ is a valid abstract skyline set based on Definition 3 because all skyline points are either dominated or belong to $\widehat{SKY'}$. This is because only point p was removed from $\widehat{SKY}$ and all points q_j dominated by p are dominated by q. Furthermore, it holds that p dominates q because p dominates q_j, $\forall j$, therefore $p[i] \leq q_j[i]$, $\forall d_i \in D$, leading to $p[i] \leq min_{\forall q_j}(q_j[i]) = q[i]$, $\forall d_i \in D$. Based on Definition 4 it holds that $\widehat{SKY'} \preceq \widehat{SKY}$.

The previous theorem states that even though there exist infinite abstract skyline sets, only a fraction of those, namely those that have each coordinate equal to the coordinate of some skyline point, are candidates for data summarization of good quality. More particularly, even though an abstract skyline point can be placed anywhere in the data space as long as it dominates a skyline point, Theorem 3 shows that it only makes sense to select the data points that belong to a grid, such as the one depicted in Fig. 6(b). Reversing this observation, if a subset of the skyline points p_i that should be abstracted by one abstract skyline point p is given, then the coordinates of the abstract skyline point p are uniquely defined as $\forall d_j \in D$: $p[j] = min_{\forall p_i}(p_i[j])$.

Based on this discussion, we conclude that finding an abstract skyline set that leads to a data summarization of good quality can be done in two discrete steps. First, B groups of skyline points are found, and second each group of skyline points is replaced by an abstract skyline point. The remaining challenge is to determine appropriate groups of skyline points.

In order to create the B groups of skyline points, we quantify the induced error (cost) by using the L_∞ distance $d()$ between any two skyline points[3]. Skyline points with smaller L_∞ distance are expected to lead to an abstract skyline with smaller error regions, when replaced by one abstract skyline point. For example, in Fig. 5, grouping points p and q and replacing them with point r is worse than grouping points m and k together. Then, the problem can be expressed as finding the B skyline points p_i, such that the maximum L_∞ distance of any other skyline point p to its closest point p_i is minimized. Thus, the aim is to minimize the following cost function:

$$cost(\{p_1, \ldots, p_B\}) = max_{\forall p \in SKY} min_{\forall p_i, 1 \leq i \leq B} d(p, p_i)$$

Problem 1 (*Optimal Abstract Skyline Problem*) Given an upper limit B, find the B groups $C_1, \ldots, C_B$ of skyline points, or equivalently the B skyline points p_i such that $cost(\{p_1, \ldots, p_B\})$ is minimized. The optimal abstract skyline is defined as $\widehat{SKY} = \{q_i\}$ such that for each C_i an abstract skyline point q_i is defined as $\forall d_j \in D : q_i[j] = min_{\forall p' \in C_i}(p'[j])$.

[3] The distance $d(p, q)$ of two points p and q based on the L_∞ distance is $d(p, q) = \max_{\forall d_i}(|p[i] - q[i]|)$.

Algorithm 2 Abstract skyline creation on a server

1: **Input:** SKY, B
2: **Output:** $\widehat{SKY}$
3: $p_1 = argmin_{\forall p \in SKY}(\sum_{1 \leq i \leq d} \ln(p[i]+1))$
4: **for** $(\forall p_i : i = 2 \dots b)$ **do**
5: $\quad p_i = max_{\forall p \in SKY - \{p_1,\dots,p_{i-1}\}} dist(p, \{p_1, \dots, p_{i-1}\})$
6: **end for**
7: **for** $(\forall p \in SKY)$ **do**
8: $\quad i = argmin_{\forall p \in SKY} d(p, p_i)$
9: $\quad C_i = C_i \bigcup \{p\}$
10: **end for**
11: **for** $(1 \leq i \leq B)$ **do**
12: $\quad \forall d_j \in D : q[j] = min_{\forall p' \in C_i}(p'[j])$
13: $\quad \widehat{SKY} = \widehat{SKY} \bigcup q$
14: **end for**

The solution of the optimal abstract skyline problem is essentially the optimal solution of the k-center problem [16] applied on the skyline points based on L_∞ distance. For dimensionality at least 3, this problem is known to be NP-hard. In order to compute the optimal abstract skyline problem of a two-dimensional skyline set SKY, a dynamic programming algorithm can be applied. Nevertheless, since the skyline sets in the two dimensional space are rather smaller, we focus on the more general case of the d-dimensional problem. Thus, we propose an approximate algorithm for the d-dimensional optimal abstract skyline problem.

6.3 Abstract Skyline Creation Algorithm

We now present an approximate algorithm for finding an appropriate abstract skyline set of size B. We denote as $dist(p, \{p_1, \dots, p_j\}) = min_{\forall p_i} d(p, p_i)$ the minimum L_∞ distance ($d(p, p_i)$) between point p and any point p_i of the set $\{p_1, \dots, p_j\}$. Algorithm 2 initially picks one skyline point for creating the first group of skyline points. In this step, any point can be chosen randomly. Inspired by SFS [13], we choose the skyline point p with the smallest 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. Following a greedy approach, Algorithm 2 picks as the next point for creating a group, the skyline point that would increase mostly the L_∞ distance if it was grouped together with the previously chosen skyline point. Thus, the next point p_i is set as the point with the maximum $dist(p, \{p_1, \dots, p_{i-1}\})$. After selecting B skyline points p_i for creating the groups, each skyline point p is assigned to the group of p_i with minimum L_∞ distance ($d(p, p_i)$) between the skyline point p and point p_i. Finally, for each group C_i of skyline points, a point q with $\forall d_j \in D\ q[j] = min_{p' \in C_i}(p'[j])$, is added to the abstract skyline set $\widehat{SKY}$.

As an example, consider again the data set depicted in Fig. 6(a) and assume that B is set equal to 3. Let us assume that point c is picked as an initial point by Algorithm 2. Then according to Algorithm 2 in the next step, point e is selected since it has the highest L_∞ distance from c. In the next step, skyline point a is selected. In the next phase, all points in SKY are assigned to one of the groups. Thus, b is assigned to a and d to e based on the L_∞ distance. Finally, the resulting abstract skyline set is defined as $\widehat{SKY} = \{p, c, r\}$. Recall that based on Lemma 4, any query line meets one of the abstract skyline points p, c or r before any skyline point. Moreover, the abstract skyline points bound as tightly as possible the skyline points, which enables effective server selection during query processing.

6.4 Adapting DiTo

Algorithm 1 needs only minor modifications to use the abstract skyline set as a data summarization and still provide the correct result. The required modification is related to the threshold value. In order to ensure that at least k objects exist with better score than the threshold, the threshold must be set based on the k best data points retrieved so far, and not based on scores of summary objects. This is because, in contrast to skyline points that are actual data points, abstract skyline points may not exist in the data set, and therefore we are no longer certain that at least k objects with smaller score exist, if summarization objects are used for setting the threshold. Obviously, the derived threshold values by using the abstract skyline set are more loose, leading to more transferred data during query processing.

On the other hand, the benefit of using the abstract skyline set as a data summarization of size B is that DiTo is more scalable, since the construction and maintenance cost does not increase due to the increase of the cardinality of the skyline set. Furthermore, DiTo becomes more robust to data updates when the abstract skyline is used as a data summarization. DiTo can guarantee accurate query results in the case of data updates, if the summarization objects of the data set do not change. Otherwise, the new summarization objects have to be sent to S_C. An important property of the abstract skyline set is that it constitutes a lower bound of the skyline set and is less likely to change than the original skyline set. In the case of high update rate, the abstract skyline may be appropriate, in order to reduce the maintenance cost and shorten the time intervals where DiTo cannot provide exact query results.

The following lemma shows that the correctness of DiTo is not violated, when the abstract skyline is used as data summarization, instead of the skyline set.

Lemma 5 *(Correctness of DiTo using abstract skyline) Let TOP_k be the set of points returned by DiTo as the top-k result set for a given scoring function f, when the abstract skyline is used. Then, it holds that for any $q \in TOP_k$: $\nexists p \in O_i$ such that $p \notin TOP_k$ and $f(p) < f(q)$.*

Proof (*Sketch*) Let us assume that $\exists q \in TOP_k$ and $\exists p \in O_i$ such that $p \notin TOP_k$ and $f(p) < f(q)$. We distinguish two cases. First, if $p \in \widehat{SKY}_i$, and since $f(p) < f(q)$, we derive that p is accessed before q because DiTo accesses points p' in increasing order of $f(p')$. Therefore, $p \in TOP_k$, which is a contradiction. Secondly, if $p \notin \widehat{SKY}_i$, then p is dominated by at least one point $p' \in \widehat{SKY}_i$. Due to Lemma 1, it holds that $f(p') \leq f(p)$, hence $f(p') < f(q)$. This in turn means that the top-k list of server S_i is retrieved, and therefore also point p, and thus $p \in TOP_k$, which is a contradiction.

7 Maintenance Issues

DiTO relies on summary information that is collected prior to the actual query processing. The skyline information is published to the coordinator and serves as data summarization. In the case of a server joining the network, its skyline needs to be published to the coordinator. When a server leaves the network, it is detected by lack of response during query processing. When this occurs, the summary entry of the departed server is removed at the coordinator. DiTO reports the correct and complete result set under the assumption of a system with static data.

In this section, we discuss the case of dynamic data and how they influence the maintenance of the summary information. We focus on two basic operations: insertions of new points and deletions of existing points. Notice that an update can be modeled as a deletion followed by an insertion. As will be demonstrated also experimentally, when additions of new data follow the data distribution, the skyline of the data set rarely changes. In general, insertions or deletions of points p at server S_i that do not belong to the skyline set SKY_i can be safely ignored, as they do not affect DiTo also for the case of abstract skyline. Hence, no maintenance operation is necessary for such points. Next, we focus on the case of insertions and deletions of skyline points $p \in SKY_i$. The following discussion separates the way maintenance is performed in the case of skyline and abstract skyline respectively.

Maintenance of Skyline. In the case of insertion of a new point p, if p is a skyline point, then the skyline set SKY_i needs to be updated. In particular, p is added to the skyline set and those points dominated by p are removed. The new skyline set SKY_i' needs to be sent to the coordinator server S_C, in order to ensure the correctness of DiTo during query processing. In the case that a point $p \in SKY_i$ is deleted, then the skyline set needs to be updated. The new skyline set SKY_i' should be sent to S_C, and then the correctness of DiTo is again guaranteed.

Maintenance of Abstract Skyline. In the case of insertion of a point p at server S_i that causes an update of SKY_i, we first need to examine if there exists an abstract skyline point p' that dominates p, i.e., $p' \in \widehat{SKY}_i$ and p' dominates p. If such a point does not exist, then we cannot avoid recomputing the abstract skyline and sending it to S_C. If there exists such a point p', then we distinguish two strategies for maintenance. The first strategy simply

recomputes the abstract skyline, and if it has changed, then it is sent to S_C. This strategy ensures that S_C always keeps an abstract skyline of S_i that is a solution to the optimal abstract skyline problem. The second strategy follows a lazy approach that does not update S_C, since the correctness of DiTo is guaranteed due to the existence of point p' that dominates p.

In the case of deletion of a skyline point p, we can have two strategies, as above. Either the abstract skyline set is recomputed and sent to S_C, or nothing happens in which case the correctness of DiTo is still guaranteed, at the expense of not having an abstract skyline $\widehat{SKY}_i$ at S_C that is a solution to the optimal abstract skyline problem.

8 Experimental Evaluation

In this section, we evaluate experimentally the performance of DiTo. DiTo was implemented in Java[4], while the network aspects were simulated, in order to study the scalability of our approach. All experiments were conducted on a machine with 2x4 cores (AMD Opteron), 32GB RAM, and 2TB HDD.

Parameter	**Values**
Dimensionality	2, **4**, 6, 8, 10
Cardinality of data set	**10M**, 100M
Data distributions	**UN**, CL
Number of servers	**2K**, 4K, 6K, 8K
Value of k	**10**, 20, 30, 40, 50

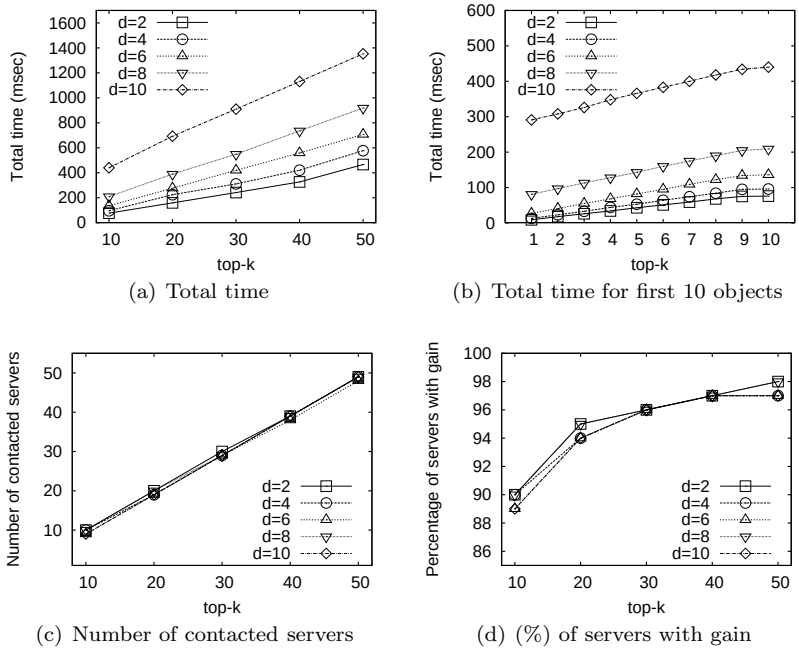
Table 2 Experimental parameters and values

8.1 Experimental Setup

In our experimental study, we used synthetic (uniform and clustered) data sets. In all experiments, the data set is horizontally partitioned to servers evenly. The uniform data set includes random points in a space $[0, L]^d$, where d denotes the dimensionality. For the clustered data set, each server picks 10 cluster centroids randomly and the coordinates of all generated points follow a Gaussian distribution on each axis with variance 0.25, and a mean equal to the corresponding coordinate of the centroid. We conduct experiments varying the dimensionality d (2-10), the cardinality n (10M-100M) of the data set, the number of servers N (2K-8K), and the value k of the top-k queries. The parameters and values used in our experiments are outlined in Table 2 (the values in bold are the default values).

For each experiment, we generate 20 queries with random weightings and report the average values obtained. We measure the average: (i) total response

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



(a) Total time (b) Total time for first 10 objects

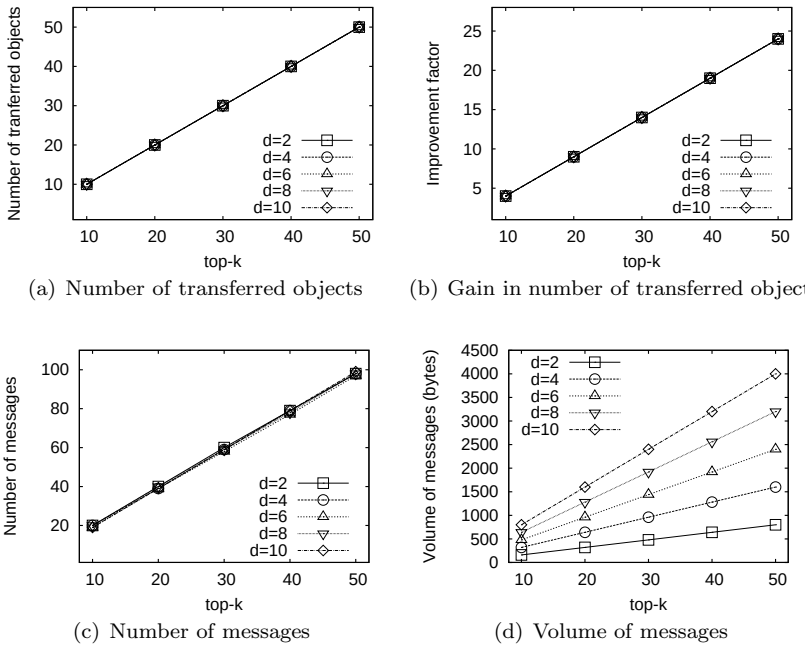
(c) Number of contacted servers (d) (%) of servers with gain

Fig. 7 Performance (time and contacted servers) of DiTo for varying dimensionality and top-k values

time (including network delay), (ii) number of contacted servers, (iii) volume of transferred data, (iv) number of transferred objects, and (v) number of messages. To simulate the network delay, we assume 50KB/sec as network transfer rate on connections between servers. We also varied the network transfer rate, but the obtained results were always qualitatively similar, thus they are not reported here. The response time is measured as the sum of processing time and network transfer time required for the objects transferred in the network. For the underlying local top-k query processing, DiTo employs a state-of-the-art branch-and-bound top-k algorithm that uses an R-tree [28].

For comparison purposes, we implemented the BRANCA algorithm [36], which is the approach closest to DiTo and is shown to outperform earlier approaches, such as [4]. In BRANCA, the servers form an unstructured peer-to-peer topology, and caching of previous top-k results is employed on multiple servers as the results are propagated in the network, in order to prune complete routing paths when future queries arrive. We used the GT-ITM topology generator [5] to create well-connected random graph topology of 2K servers with an average connectivity degree of 6. We used half of the queries to warm-up the cache, we employed a cache of size 50 entries, and cache replacement is triggered when 80% of the cache is full.

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



(a) Number of transferred objects (b) Gain in number of transferred objects

(c) Number of messages (d) Volume of messages

Fig. 8 Performance (transferred data and messages) of DiTo for varying dimensionality and top-k values

8.2 Performance Analysis of DiTo

In Fig. 7 and Fig. 8, we study the performance of DiTo for varying dimensionality and also for variable k values. We increase the dimensionality of the data set by 2 starting from 2 to 10, and the value of k by 10 starting from 10 to 50, thus a total of 25 (=5 · 5) experimental setups is tested. For the remaining parameters, the default setup is employed; the data set is uniform, its cardinality is 10M, and the number of servers is 2K.

Fig. 7(a) shows the total time for reporting the top-k result set, including both processing time and network transfer time. As shown in the chart, the total time is always less than 1.5 seconds, even for the 10-dimensional data set and k=50. When k increases, the total time also increases because more processing is required to report more result objects. When the dimensionality increases, the total time increases both because the processing cost is higher for higher dimensions, as well as due to the higher network transfer cost due to objects of larger size (i.e., represented by more dimensions). It is also noteworthy that the total time is dominated by the processing time, because the network transfer time is very small. The reason is that the number k of objects that need to be reported is small (between 10 and 50), which can be delivered very fast even when the network transfer rate is small.

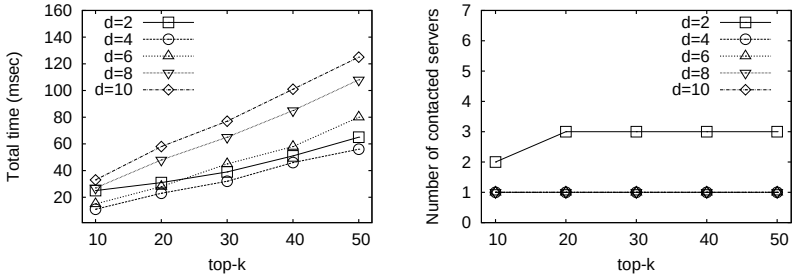
In Fig. 7(b), we demonstrate the progressiveness of DiTo in terms of total time for reporting the first 10 objects. It is obvious that DiTo can report results progressively, requiring only few milliseconds to report the first result and only marginal extra time for the subsequent results.

In the next chart in Fig. 7(c), the number of contacted servers for computing the top-k results is shown. Regardless of the dimensionality, the chart shows that k servers are contacted for reporting the top-k result. The reason is that the data set is uniform therefore the top-k results are probably stored at k different servers. DiTo exploits the collected skyline sets to minimize the number of servers that need to be contacted to produce the correct result. This is a strong feature of DiTo, which ultimately leads to good performance. As a complementary result, in Fig. 7(d), we show the percentage of servers out of the contacted servers, which managed to avoid transferring some objects due to the use of threshold. We refer to such servers as servers with gain. The result shows that in all setups more than 88% of the servers achieve some gain.

Fig. 8(a) shows the number of transferred objects for reporting the top-k result. The chart shows that in all cases only k objects need to be transferred, namely the top-k objects. This shows that DiTo eagerly avoids transferring unnecessary objects during query processing. This is strong evidence regarding the effectiveness of the threshold employed in DiTo. To demonstrate this effect more clearly, we report in Fig. 8(b) the improvement factor in terms of transferred objects, defined as the ratio of the number of objects that would have been additionally transferred without using the threshold over the number of objects that were actually transferred. For example, for d=4 and top-k=50, the improvement factor is 24.3, computed as the ratio of 1215 objects that would have been additionally transferred over 50.

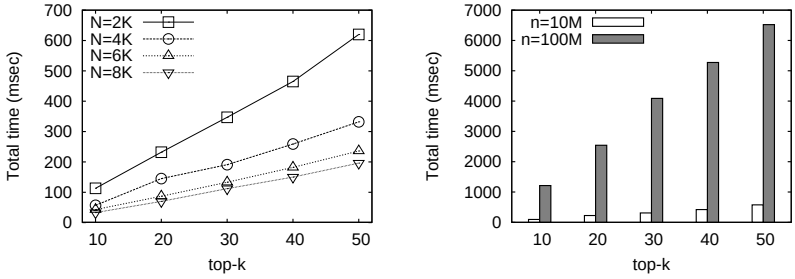
Fig. 8(c) reports that number of messages for computing the top-k results. For the uniform data set used, the number of required messages is always $2{\cdot}k$ regardless of the dimensionality. This relates to the fact that k different servers contain the top-k results, thus for each server 2 messages are necessary; one to contact the server and another one to collect its result. Fig. 8(d) shows the volume of the messages, measured in bytes. When the dimensionality is higher, the objects are represented by more dimensions, thus using more bytes, which explains the increasing tendency as the dimensionality increases.

Furthermore, we study the performance of DiTo for the clustered data set in Fig. 9. Intuitively, when the data set is clustered, DiTo is more efficient than in the case of uniform data, because multiple top-k results are usually located on the same server. In Fig. 9(a), the total time is depicted for varying dimensionality and top-k values. Compared to Fig. 7(a) which corresponds to the same experimental setup but for uniform data set, we observe that the total time is much smaller, often by one order of magnitude. This verifies the intuition about the case of clustered data. Moreover, we report in Fig. 9(b) the number of contacted servers for the same experiment. Clearly, in all cases, the top-k results are located on few servers, which explains the good performance of DiTo in the case of clustered data. In the following, we show the results



(a) Total time (b) Number of contacted servers

Fig. 9 Performance (time and contacted servers) of DiTo for clustered data set, and for varying dimensionality and top-k values



(a) Total time vs. number of servers (b) Total time vs. cardinality

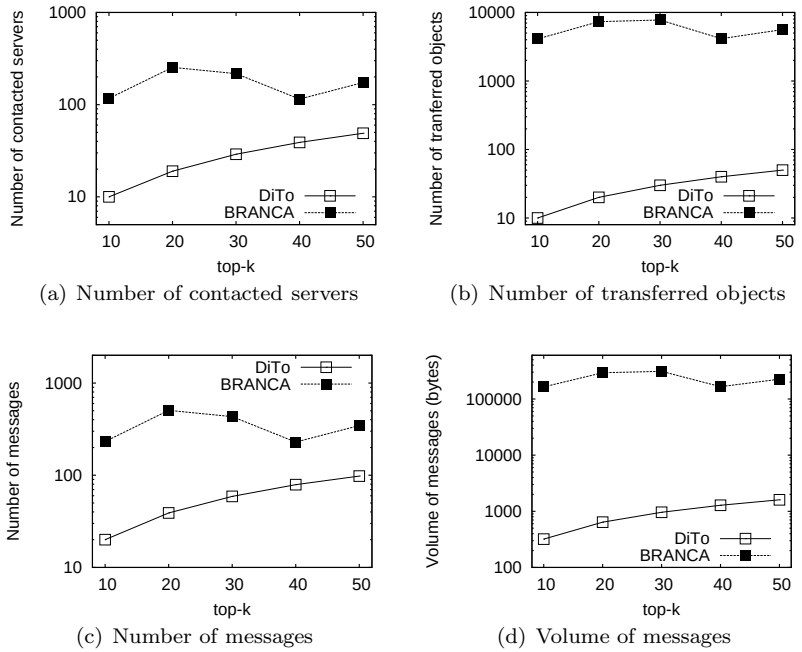
Fig. 10 Scalability study for varying number of servers and cardinality of data set

on uniform data sets only, since they constitute a harder setup than clustered data for top-k processing.

8.3 Scalability Study

In order to study the scalability of DiTo, we performed experiments for increasing values of servers as well as for higher cardinality values of the data set. The obtained results are reported in Fig. 10.

First, in Fig. 10(a), we study the performance of DiTo in terms of total time for higher number of servers, up to 8K servers. The chart shows that when the number of servers is higher, the top-k results are reported faster. This is due to the fact that the total time is dominated by the processing costs and although the number of servers is increased the cardinality of the complete data set is constant. Thus, each server stores fewer data points, thereby requiring smaller processing time. This explains the decreasing tendency of total time as the number of servers increases. All in all, the performance of DiTo is not affected by increasing the number of servers.



(a) Number of contacted servers (b) Number of transferred objects

(c) Number of messages (d) Volume of messages

Fig. 11 Comparison of performance of DiTo to the BRANCA algorithm

Then, in Fig. 10(b), we keep the number of servers fixed and equal to 2K and increase the data set size by one order of magnitude, from 10M to 100M data points. In this case, the processing cost at each individual server increases, due to the increase in the local data set size by a factor of 10. In consequence, the total time also increases with the cardinality of the data set. Still, even for the most demanding setup of 100M data points, DiTo reports the top-10 results in a little bit over one second, and the top-50 result in 6.5 seconds, which is acceptable given the high cardinality of the data set and the high degree of distribution.

8.4 Comparison to BRANCA

In addition, in Fig. 11, we provide a comparison of DiTo to the BRANCA algorithm proposed in [36]. BRANCA aggressively stores the results of previous top-k queries in caches of intermediate servers (as long as the cache size permits it) on the routing paths used to report the result to the querying server. If a cache becomes full, an appropriate cache replacement algorithm is invoked to keep only the most useful entries. Then, when a new top-k query is processed, each cache is exploited to provide an answer to the query from the previous locally cached results, without having to propagate the query further in the routing path, thereby saving communication cost.

We use the default setup of a 4-dimensional data set of cardinality 10M that follows a uniform distribution, the number of servers is 2000, and we vary the value of k. Notice that the y-axis is in log-scale in all charts.

In Fig. 11(a), we show the number of contacted serves by both algorithms. Obviously, BRANCA needs to contact many more servers to report the correct top-k result, despite the use of caches at all servers. Especially in the case of top-10 queries, DiTo requires one order of magnitude fewer servers to be contacted than BRANCA, which demonstrates the nice scaling features of DiTo, considering the case of multiple top-k queries being issued at the same time from different servers.

Fig. 11(b) shows the number of transferred objects by each algorithm. Again, DiTo is much more efficient than BRANCA, transferring fewer objects by 2 orders of magnitude. DiTo exploits the data summarizations to deliberately retrieve carefully selected objects by servers, whereas BRANCA requires remote servers to report their local top-k result sets, even though these objects may not belong to the top-k result set eventually.

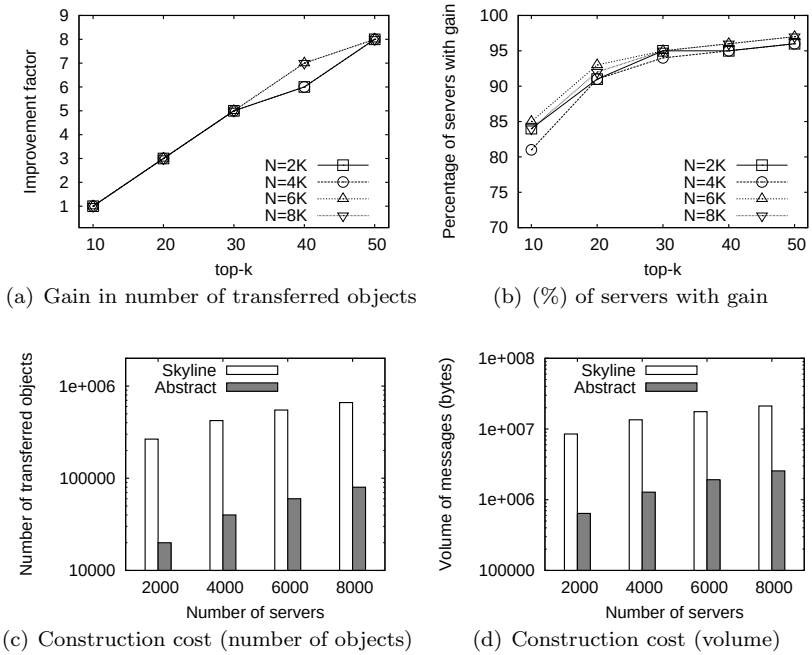
Fig. 11(c) depicts the number of messages required by the two algorithms compared. This experiment closely follows the result in terms of number of contacted servers shown in Fig. 11(a). DiTo always uses significantly fewer messages, as fewer servers need to be contacted during query processing to compute the top-k result set. Last but not least, in Fig. 11(d), the volume consumed by transferred objects is shown. Again, DiTo is much more efficient than BRANCA, and requires much fewer bytes to be transmitted to report the top-k result.

8.5 Abstract Skyline

In Fig. 12, we evaluate the performance of DiTo for the case where the abstract skyline is used as a data summarization. In this experiment, the size of the abstract skyline set is bounded to 10 points for each server. We use the default setup and vary the number of servers from 2K to 8K.

Fig. 12(a) shows the improvement impact factor for the case that the abstract skyline is used as data summarization. The chart shows that even when using a small summary of 10 abstract skyline points, the improvement factor is still high and up to 8, regardless of the number of servers. This is strong evidence that the performance of query processing is still efficient in the case of abstract skyline.

In addition, we provide in Fig. 12(b) the percentage of servers with gain when the abstract skyline is used. The results clearly show that in all cases more than 80% of the servers manage to avoid transferring some object by exploiting the threshold. Compared to the lower bound percentage of 88% for the experiment when the skyline set is employed (Fig. 8(b)), we conclude that the use of the abstract skyline set maintains practically all the benefits of the skyline set, in terms of servers with gain. Moreover, the abstract skyline



(a) Gain in number of transferred objects (b) (%) of servers with gain

(c) Construction cost (number of objects) (d) Construction cost (volume)

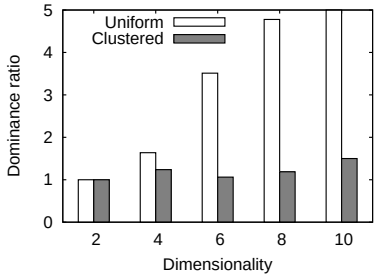
Fig. 12 Experiments with abstract skyline of size 10 points

induces much smaller construction and maintenance cost due to its smaller size.

This latter fact is demonstrated in the next chart in Fig. 12(c), where the construction cost of the abstract skyline set is compared to the cost of skyline set, in terms of number of transferred summary objects (abstract skyline and skyline respectively) from the servers to the coordinator, for varying the number of servers. Notice that the y-axis is in log-scale. The use of abstract skyline set of size 10 points achieves a reduction in the number of objects transferred for construction by one order of magnitude. This important benefit comes without deteriorating the efficiency of query processing, as demonstrated earlier.

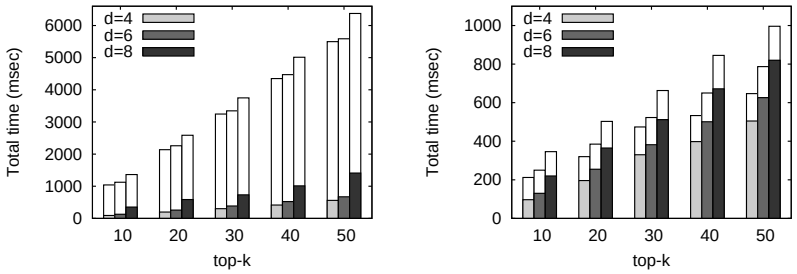
In Fig. 12(d), we show the construction cost in terms of volume of messages (measured in bytes) for sending the summaries to the coordinator. Again, using the abstract skyline set achieves significant reduction in the transferred volume, thereby decreasing the construction cost.

In Fig. 13, we demonstrate the quality of the abstract skyline by means of an intuitive measure. We define the *Dominance ratio* as the average number of abstract skyline points that dominate a skyline point. Small values of dominance ratio are preferable, since they indicate that each abstract skyline point dominates only few skyline points. Ideally, this ratio would be equal to one, in which case each abstract skyline point dominates a single skyline point. In the



(a) Dominance ratio

Fig. 13 Experiments with dominance ratio for abstract skyline of size 10 points



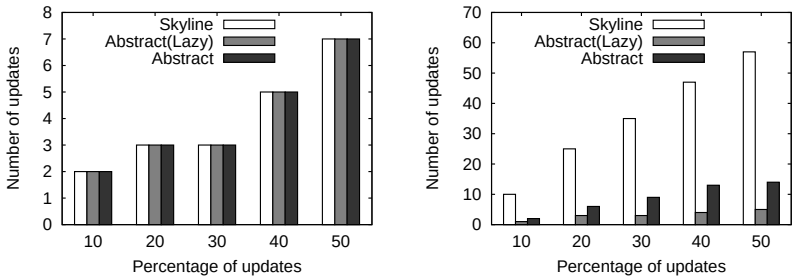
(a) DiTo (b) Parallel variant of DiTo

Fig. 14 Experiments with latency

chart of Fig. 13, we measure the dominance ratio in the case of uniform and clustered data on a random server with 10K data points for varying dimensionality. The values of the dominance ratio are always very small regardless of dimensionality, and this is a strong argument in favor of the quality of the abstract skyline. More importantly, these small values are produced even though the number of skyline points is high, e.g. for $d = 6$, the number of skyline points is 963 and each is dominated (on average) only by 3.5 abstract skyline points.

8.6 Effect of Latency and Parallel Variant of DiTo

In Fig. 14, we demonstrate the performance of the parallel variant of DiTo. To show its merit, we perform an experiment in which each communication between two servers incurs a latency of 100 msec. We compare the classic variant of DiTo (Fig. 14(a)) that contacts servers sequentially against the parallel variant of DiTo (Fig. 14(b)). We use the values 4,6, and 8 for dimensionality, which correspond to the three bars for each value of k in the chart. The colored part of the bars corresponds to the processing time, while the white extension corresponds to the network time (transfer of data and latency).



(a) Dimensionality d=2 (b) Dimensionality d=4

Fig. 15 Experiments with data updates

As expected, when the parallel variant of DiTo is employed, the total time is significantly reduced. More importantly, this is the result of reducing the white part of the bars, which means that even when the latency is explicitly taken into account, the parallel variant of DiTo diminishes the time spent for networking (transfer and latency). Thus, DiTo is very efficient in terms of performance, regardless of the effect of latency.

8.7 Data Updates

In Fig. 15, we study the effect of data updates in the skyline and abstract skyline sets for dimensionality values 2 and 4. In more detail, we pick a random server that stores 10K data points, and we gradually add new data points that follow the data distribution on the server, from 10% (1K points) to 50% (5K points) of the server's data. We measure how many updates in the skyline and abstract skyline sets are triggered due to the updates of data. In particular, we measure (i) the number of updates in the skyline set (denoted *Skyline*), (ii) the number of updates in the abstract skyline set caused by an update in the skyline set (denoted *Abstract*), and (iii) the number of updates in the abstract skyline set caused only when the data update affects the correctness of the abstract skyline (denoted *Abstract(Lazy)*). When the dimensionality is small (d=2 in Fig. 15(a)) the number of updates triggered are minimal and always fewer than 7 (out of 5K data updates). In addition, even for d=4 (Fig. 15(b)), the number of updates triggered in the skyline set are around 1% of the data updates. For the abstract skyline this number is even smaller, i.e., it rarely needs to be updated. As expected, the approach with lazy updates of the abstract skyline set requires very few updates and still guarantees correctness.

9 Conclusions

In this paper, we study the challenging problem of efficient top-k query processing processing over multiple servers, where each server stores autonomously a

fraction of the data. Our approach, called DiTo, relies on a threshold-based algorithm which forwards the top-k query only to the servers that store relevant data, in such a way that the amount of transferred data is minimized. DiTo always returns the correct result set for any top-k query, while supporting a large class of scoring functions. Furthermore, we study the problem of bounding the cardinality of the data summarization used for the server selection process. Finally, our experimental evaluation demonstrates the feasibility of our approach.

References

1. Akbarinia, R., Pacitti, E., Valduriez, P.: Reducing network traffic in unstructured P2P systems using top-k queries. Distributed and Parallel Databases **19**(2-3), 67–86 (2006)
2. Akbarinia, R., Pacitti, E., Valduriez, P.: Best position algorithms for top-k queries. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 495–506 (2007)
3. Balke, W.T., Güntzer, U.: Multi-objective query processing for database systems. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 936–947 (2004)
4. Balke, W.T., Nejdl, W., Siberski, W., Thaden, U.: Progressive distributed top-k retrieval in peer-to-peer networks. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 174–185 (2005)
5. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 421–430 (2001)
6. Bruno, N., Chaudhuri, S., Gravano, L.: Top-k selection queries over relational databases: Mapping strategies and performance evaluation. ACM Trans. Database Syst. **27**(2), 153–187 (2002)
7. Cao, P., Wang, Z.: Efficient top-k query calculation in distributed networks. In: Proceedings of Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 206–215 (2004)
8. Chang, Y.-C., Bergman, L.D., Castelli, V., Li, C.-S., Lo, M.-L., Smith, J.R.: The Onion Technique: Indexing for Linear Optimization Queries. In: Proceedings of ACM International Conference on Management of Data (SIGMOD), pp. 391–402 (2000)
9. Chaudhuri, S., Dalvi, N.N., Kaushik, R.: Robust cardinality and cost estimation for skyline operator. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), p. 64 (2006)
10. Chaudhuri, S., Gravano, L.: Evaluating top-k selection queries. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 397–410 (1999)
11. Chaudhuri, S., Gravano, L., Marian, A.: Optimizing top-k selection queries over multimedia repositories. IEEE Trans. Knowl. Data Eng. **16**(8), 992–1009 (2004). DOI http://doi.ieeecomputersociety.org/10.1109/TKDE.2004.30
12. Chen, C.M., Ling, Y.: A sampling-based estimator for top-k selection query. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 617–627 (2002)
13. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 717–816 (2003)
14. Dedzoe, W.K., Lamarre, P., Akbarinia, R., Valduriez, P.: ASAP top-k query processing in unstructured P2P systems. In: Proceedings of International Conference on Peer-to-Peer Computing (P2P), pp. 1–10 (2010)
15. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proceedings of Symposium on Principles of Database Systems (PODS), pp. 102–113 (2001)
16. Gonzalez, T.F.: Clustering to minimize the maximum intercluster distance. Theor. Comput. Sci. **38**, 293–306 (1985)

17. Güntzer, U., Balke, W.T., Kießling, W.: Optimizing multi-feature queries for image databases. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 419–428 (2000)
18. Hose, K., Karnstedt, M., Sattler, K.U., Zinn, D.: Processing top-N queries in P2P-based web integration systems with probabilistic guarantees. In: Proceedings of International Workshop on Web and Databases (WebDB), pp. 109–114 (2005)
19. Hristidis, V., Koudas, N., Papakonstantinou, Y.: PREFER: A system for the efficient execution of multi-parametric ranked queries. In: Proceedings of ACM International Conference on Management of Data (SIGMOD), pp. 259–270 (2001)
20. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K., Elmongui, H.G., Shah, R., Vitter, J.S.: Adaptive rank-aware query optimization in relational databases. ACM Trans. Database Syst. **31**(4), 1257–1304 (2006)
21. Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-k query processing techniques in relational database systems. ACM Computing Surveys **40**(4) (2008)
22. Lu, J., Callan, J.: Merging retrieval results in hierarchical peer-to-peer networks. In: Proceedings of the ACM International Conference on Research and Development in Information Retrieval (SIGIR), pp. 472–473 (2004)
23. Lu, J., Callan, J.: Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In: Proceedings of European Conference on IR Research (ECIR), pp. 52–66 (2005)
24. Marian, A., Bruno, N., Gravano, L.: Evaluating top-k queries over web-accessible databases. ACM Trans. Database Syst. **29**(2), 319–362 (2004)
25. Michel, S., Triantafillou, P., Weikum, G.: KLEE: a framework for distributed top-k query algorithms. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 637–648 (2005)
26. Mouratidis, K., Bakiras, S., Papadias, D.: Continuous monitoring of top-k queries over sliding windows. In: Proceedings of ACM International Conference on Management of Data (SIGMOD), pp. 635–646 (2006)
27. Ryeng, N.H., Vlachou, A., Doulkeridis, C., Nørvåg, K.: Efficient distributed top-k query processing with caching. In: Proceedings of DASFAA (2), pp. 280–295 (2011)
28. Tao, Y., Hristidis, V., Papadias, D., Papakonstantinou, Y.: Branch-and-bound processing of ranked queries. Information Systems **32**(3), 424–445 (2007)
29. Tsaparas, P., Palpanas, T., Kotidis, Y., Koudas, N., Srivastava, D.: Ranked join indices. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 277–288 (2003)
30. Vlachou, A., Doulkeridis, C., Kotidis, Y., Nørvåg, K.: Reverse top-k queries. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 365–376 (2010)
31. Vlachou, A., Doulkeridis, C., Kotidis, Y., Nørvåg, K.: Monochromatic and bichromatic reverse top-k queries. IEEE Trans. Knowl. Data Eng. **23**(8), 1215–1229 (2011)
32. Vlachou, A., Doulkeridis, C., Nørvåg, K.: Monitoring reverse top-k queries over mobile devices. In: Proceedings of ACM Workshop on Data Engineering for Wireless and Mobile Access (MobiDE) (2011)
33. Vlachou, A., Doulkeridis, C., Nørvåg, K., Kotidis, Y.: Identifying the most influential data objects with reverse top-k queries. PVLDB **3**(1), 364–372 (2010)
34. Vlachou, A., Doulkeridis, C., Nørvåg, K., Vazirgiannis, M.: On efficient top-k query processing in highly distributed environments. In: Proceedings of ACM International Conference on Management of Data (SIGMOD), pp. 753–764 (2008)
35. Vlachou, A., Doulkeridis, C., Nørvåg, K., Vazirgiannis, M.: Skyline-based peer-to-peer top-k query processing. In: Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 1421–1423 (2008)
36. Zhao, K., Tao, Y., Zhou, S.: Efficient top-k processing in large-scaled distributed environments. Data and Knowledge Engineering **63**(2), 315–335 (2007)
37. Zou, L., Chen, L.: Pareto-based dominant graph: An efficient indexing structure to answer top-k queries. IEEE Trans. Knowl. Data Eng. **23**(5), 727–741 (2011)