J. Parallel Distrib. Comput. 71 (2011) 302–315

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Top-*k* vectorial aggregation queries in a distributed environment

Guy Sagy [a], Izchak Sharfman [a], Daniel Keren [b,*], Assaf Schuster [a]

[a] CS Faculty, Technion, Technion City 32000, Haifa, Israel
[b] CS Department, Haifa University, Haifa 31905, Israel

ARTICLE INFO

Article history:
Received 22 January 2010
Received in revised form
29 August 2010
Accepted 2 September 2010
Available online 19 September 2010

Keywords:
Top-*k* distributed algorithm
Vectorial aggregation
Geometric method

ABSTRACT

Given a large set of objects in a distributed database, the goal of a top-*k* query is to determine the top-*k* scoring objects and return them to the user. Efficient top-*k* ranking over distributed databases has been the focus of recent research, with most current algorithms operating on the assumption that each node holds a single or small subset of each object's numerical attributes. However, in many important setups each node might hold instead a full *d*-dimensional vector of numerical attributes for each object. Examples include website activity in distributed servers, sales statistics for a retail chain, or share price information in different stock markets. For these setups, we define a novel ranking problem, *top-k vectorial aggregation queries*, where each object's score is determined by first aggregating the attribute vectors held for it and then applying the scoring function over the aggregated vector.

Our communication-efficient algorithm uses a blend of geometric and skyline related machinery, some of which is newly developed, as well as an algorithmic framework for defining generic local constraints. Whereas previous algorithms have reduced data sharing by defining local thresholds for each attribute, such tailored solutions might perform poorly. Experimental results on real-world data demonstrate that our algorithm maintains low latency, with a communication cost up to four orders of magnitude lower than that of existing solutions.

1. Introduction

As computer systems and data increase in size, algorithms that can help the user to efficiently navigate the vast stores of available information are more important than ever. A frequent task is to rank the information retrieved by the end user. One way to support this task is to return only the most important (top-*k*) relevant answers to a query, known as *top-k queries*. The goal of top-*k* queries is to detect the *k* highest scoring objects, according to some scoring function. A naive approach is to compute the score for all objects; however this suffers from huge communication, computation, and access costs, especially in distributed setups.

In many distributed setups, data is partitioned over distributed nodes using the same set of attributes across all nodes. Examples include sensor systems (e.g. NASA EOS [36]), retail databases (e.g. Wal-Mart [36]) and web applications (e.g. Google [23]). In these setups, object data can be stored in various nodes and the scoring function is commonly the aggregation (or sum) of the objects' local scores. Such queries are referred to *top-k aggregation queries*.

The problem of top-*k* aggregation queries have been intensively researched. A prominent family of algorithms are threshold algorithms (TA) [15,1,18,31]. These algorithms efficiently reduce the number of accessed objects by iteratively accessing the objects according to their local score, while in each iteration the maximal value of each accessed object is estimated, and some of the objects (those which are guaranteed not to be in the top-*k*) pruned. This iterative approach has been used in many top-*k* algorithms [4,19,15,9,45,21,22]. However, while these algorithms minimize the number of accessed objects, they require many iterations, which increases communication bandwidth. Recently, a three-phase algorithm, (TPUT)[8], was proposed for performing distributed top-*k* queries. The advantage of TPUT is that each phase consists of a single round-trip interaction between the nodes and a coordinator. KLEE [29] and other algorithms [52,33] use the "three-phase" technique but consider the distribution of the values at each node to define tighter local thresholds.

All these algorithms assume the aggregation is performed over local scalar (1-dimensional) values. An interesting set of top-*k* queries are those where the local scores are *d-dimensional* vectors, and the scoring function is applied over the aggregation of these vectors. For example:

Example 1. A large set of proxy servers storing the last day's statistics for website activity (Fig. 1). Each proxy stores the same

* Corresponding author.
E-mail addresses: guysagy@cs.technion.ac.il (G. Sagy), tsachis@cs.technion.ac.il (I. Sharfman), dkeren@cs.haifa.ac.il (D. Keren), assaf@cs.technion.ac.il (A. Schuster).

0743-7315/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2010.09.002

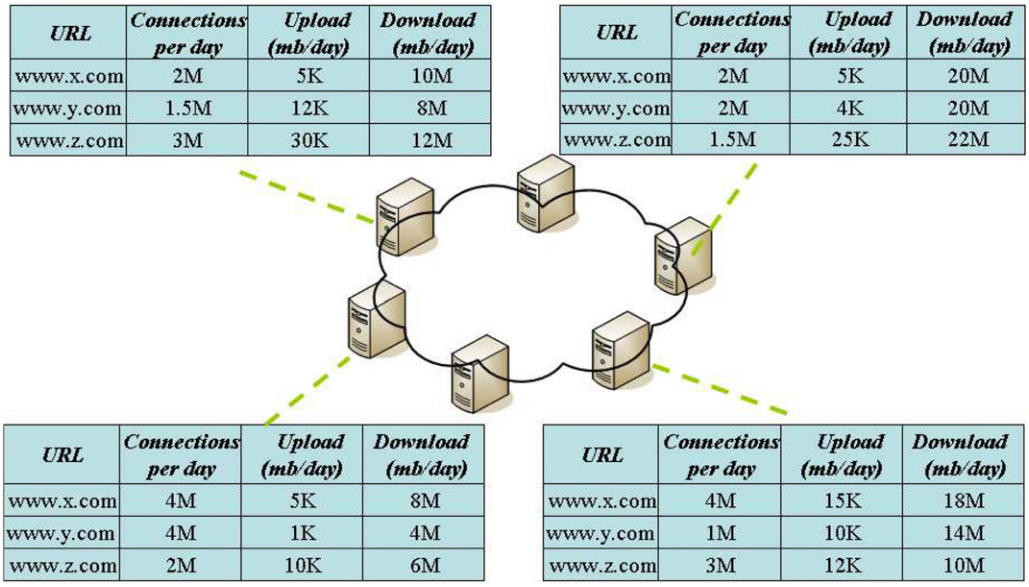


Fig. 1. Example of a distributed database, where each URL's d-dimensional vectors of attributes (connections per day, uploaded data size, downloaded data size) are stored over various routers.

Table 1
A simple example of top-1 vectorial aggregation query ($f(x, y) = xy$).

Node_1	x_1	y_1	$f(x_1, y_1)$	Node_2	x_2	y_2	$f(x_2, y_2)$	Global	$\frac{x_1+x_2}{2}$	$\frac{y_1+y_2}{2}$	$f\left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right)$
o_2	5.1	1	5.1	o_1	3	7	21	o_3	3.5	3.5	12.25
o_1	1	5	5	o_2	6.9	3	20.7	o_1	2	6	12
o_3	4	1	4	o_3	3	6	18	o_2	6	2	12

vector of attributes for each website: the number of connections, the uploaded data size, and the downloaded data size. The network manager might want to determine the top-k sites by computing the upload/download ratio, using the vectors' average values (i.e., the k sites with the highest total uploaded data size divided by the total downloaded data size).

Example 2. Consider a retailer who owns a chain of stores. The retailer wants to know which pair of products are globally most highly correlated—i.e., the correlation is computed over the global vector which equals the average of objects' local vectors, and not locally in each store. Note that the global correlation coefficient is **not** equal to the average of the local correlation coefficients, hence first the local vectors representing the sales' amounts should be aggregated, and only then the correlation coefficients computed.

In these cases and others, the score for each object is determined by first aggregating the attribute vectors held for it in the distinct nodes, and then applying the scoring function over the aggregated vector. Top-k vectorial aggregation queries can be used in many data mining applications. Other examples include finding top performing shares computed from attribute vectors from different stock markets, determining the most highly correlated query terms from different search engine query logs, and identifying denial of service attacks in distributed environments (using the same type of indicators in each router) [20].

Performing top-k vectorial aggregation queries can be challenging. As an example, look at the data in Table 1. Assume the query is to find the top-1 object using the scoring function $f(x, y) = xy$. While the local top-1 objects in node_1 and node_2 are objects o_2 and o_1 respectively, the global top-1 object is o_3. Actually, while o_3 is the *lowest* scoring object in both nodes, its global value is the *highest*. This example vividly demonstrates how difficult it is to reach correct global decisions when aggregating vectors.

In this paper we present what constitutes, to the best of our knowledge, the first treatment of the problem of *top-k vectorial aggregation queries over distributed databases*. The suggested algorithm operates in four phases, each consisting of a single round-trip interaction between the nodes and a coordinator. It uses a blend of geometric and skyline related machinery, some of which is newly developed, as well as an algorithmic framework, to compile local constraints. An efficient technique for checking whether an object violates its local constraints is also introduced, further reducing the local computational overhead. We show that the communication cost of our algorithm is up to four orders of magnitude lower than that of existing solutions.

1.1. Problem definition and notations

We assume the system consists of n nodes, $p_1, p_2, \ldots, p_n$, where each node uses the same set of attributes. Our method is also applicable when some attributes do not appear in all nodes; simply fill in the missing attributes with the average of the existing ones. There are m objects, denoted by $o_1, o_2, \ldots, o_m$, and the data for each object is partitioned over the distinct nodes, as will next be described. The set of attributes for o_j in node p_i is a d-dimensional vector $\vec{x}_{j,i}$, referred to as o_j's *local statistic vector*. The score of an object is defined as the value of a scoring function $f() : \Re^d \to \Re$ on the aggregation (weighted sum) of the local vectors. Here we define the *global statistics vector* ($\vec{x}_j$) for the object o_j as the average of the *local statistics vectors* ($\vec{x}_{j,i}$) held for it in each node i, i.e., $\vec{x}_j = \frac{\sum_{i=1}^{n} \vec{x}_{j,i}}{n}$ (note that $\vec{x}_{j,i}$ is *not* a component of $\vec{x}_j$, but a vector of the same length as $\vec{x}_j$). The score of the object o_j is $f(\vec{x}_j)$. Given a parameter k, the goal of a top-k vectorial aggregation query is to determine the k highest scoring objects while minimizing communications and access costs.

We assume the scoring function f is monotonic. Given that a vector $\vec{v}_1$ *dominates* the vector $\vec{v}_2$ ($\vec{v}_1 \succ \vec{v}_2$), i.e., all the components of $\vec{v}_1$ are not smaller than the corresponding components of $\vec{v}_2$,

then for any monotonic function $f(), f(\vec{v}_1) \geq f(\vec{v}_2)$. The scoring function can be either monotonically increasing or monotonically decreasing in each coordinate. Without loss of generality we may assume that the scoring function is monotonically increasing in all coordinates. We also assume that the components of the local vectors are positive. Since the local statistics vectors are positive, and the global statistics vector is their average, it is easy to see that for any local statistics vector, $\vec{x}_{j,i} : \vec{x}_j \succ \frac{\vec{x}_{j,i}}{n}$. In addition, given $S \subset \{1, 2, \ldots, n\}$ (a proper subset of the numbers from 1 to n), it is easy to see that:

$$\vec{x}_j \succ \frac{\sum_{i \in S} \vec{x}_{j,i}}{n}. \quad (1)$$

We assume that the query is initiated on an additional node called the coordinator, denoted by p_0. Communication is performed solely between the coordinator and the nodes (the nodes themselves do not interact). In general, this algorithm is performed over static data (as opposed to streaming setups) and scales up to hundred of nodes (as opposed to peer-to-peer networks). In this paper we do not assume node failure.

Next we present in detail a running example which illustrates the concepts and problems addressed in this paper. Consider a distributed search engine, comprised of a distributed set of n mirrors. Each mirror stores a stream of queries, where each query consists of multiple search terms. Each mirror maintains counts for the search terms that appeared in the queries. In addition, each mirror maintains a count for each pair of search terms that appeared together in the same query. This information can be used to calculate a correlation score for the co-appearance of any pair of search terms. Given two search terms, denoted A and B, let $f_{A,i}$ and $f_{B,i}$ be the respective frequency of occurrence of A and B at the mirror p_i. By frequency of occurrence we mean the number of queries the term appeared in, divided by the total number of queries. Let $f_{C,i}$ be the number of queries received at p_i which contained both A and B, divided by the total number of queries received at p_i. We denote the statistics vector maintained by the mirror p_i for the pair A and B by $\vec{x}_{AB,i} = (f_{A,i}, f_{B,i}, f_{C,i})$. Let $\vec{x}_{AB} = (f_A, f_B, f_C)$ be the global statistics vector for the pair A and B, i.e., $\vec{x}_{AB}$ represents the statistics for the pair A and B over the union of the search queries received at the mirrors. For simplicity, we assume that each mirror receives the same number of queries; therefore, $\vec{x}_{AB}$ is the average of the vectors $\vec{x}_{AB,i},\ i = 1, \ldots, n$. A widely used measure for the correlation between the appearances of A and B is defined by the correlation coefficient ρ_{AB}, which is given by:

$$\rho_{AB}(\vec{x}_{AB}) = \frac{f_C - f_A f_B}{\sqrt{(f_A - f_A^2)(f_B - f_B^2)}}.$$

The correlation coefficient receives values in the range $[-1 \cdots 1]$. A negative score indicates that the terms tend to exclude each other, a score of zero indicates that there is no correlation between the appearance of the terms, and a positive score indicates that the terms tend to appear in the same queries. Clearly, the number of co-appearances of A and B cannot exceed the number of appearances of A or that of B, i.e., $f_C \leq \min(f_A, f_B)$. Subject to this restriction, it can be shown that the correlation coefficient monotonically decreases with f_A and f_B, and monotonically increases with f_C. We are interested in determining the k pairs of search terms which have the highest correlation score. Note that the local data set held by every node is very large, since it consists of all feature pairs, hence it is crucial to develop a communication and access efficient algorithm.

1.2. Related work

Solving distributed top-k over partitioned databases has been very well studied. In these databases, each node holds a subset of the attributes of the objects and the nodes must share information in order to compute an object's score. A prominent family of algorithms are threshold algorithms (TA) [15,1,18,31], which iteratively scans the attributes in each node, one object at a time. In each iteration the algorithm computes the current k top score objects, and define a global threshold. The algorithms stop when there are k retrieved objects whose score is above the global threshold. A set of threshold algorithms have been proposed for cases where the access to attribute list is limited to sorted access only, [4,19,15], when attribute values are determined dynamically at run time [9], and when data objects are given by a join operation [45,21,22]. While these algorithms minimize the number of accessed objects, they require a large number of iterations, which increases the communication cost.

Recently, low-latency top-k algorithms have been presented. (TPUT) [8], KLEE[29] and other [52,32] algorithms have suggested to bound the number of iterations between the nodes and the coordinator and suggested a three-phase approach. While these algorithms resemble our approach and can be used to resolve top-k vectorial aggregation queries, they require viewing the scoring function as a dn-dimensional function, which may incur high communication cost.

Previous work has also studied how core database constructs, such as selection queries and views, can be employed to efficiently execute top-k queries [11,13,51,6]. Approximation of top-k query results has been studied in [2,44,10,7], studied performing top-k queries over web-accessible databases. These solutions assume specific data models which are different from ours. The problem of top-k queries in peer-to-peer environments was researched in [3,30,46,47,28,50]. However, these setups are different from our model, and assume communication among the nodes themselves.

Top-k algorithms have also been studied in central setups, where all the data is stored in one physical location. A new approach to solve top-k queries for general scoring functions has been proposed in [49,53]. Several algorithms have been proposed for resolving top-k queries in a centralized OLAP environment [26,27,48]. A common goal of these works and ours is to minimize the number of locally accessed objects, but our work considers a distributed rather than centralized setting.

Our work utilizes domination relationships among objects to reduce the number of objects retrieved locally by each node. The domination relationship has been studied in the past in the context of the skyline operator [5]. The skyline of a set of objects consists of those that are not dominated by any other object. Several algorithms have been proposed for efficiently detecting the skyline [5,12,16,34,43]. In particular, the branch-and-bound skyline algorithm (BBS) [34] uses R-trees to efficiently detect dominating objects.

Recently, a geometric approach was introduced for defining local constraints in distributed setups [41,40]. However, this approach is suited to distributed streaming setups only, does not perform well for more than a few thousand objects, and requires accessing all the objects for every query.

2. Problem model

2.1. Review of BPA, TPUT and KLEE algorithms

Previous work most related to the problems in this paper are the BPA, TPUT and KLEE algorithms for processing top-k aggregation queries. In this section, we briefly describe these algorithms, and present a variant of them which supports top-k vectorial aggregation queries. The Best Position Algorithm (BPA) [1] is designed to process top-k queries over sorted lists. This algorithm improves the performance of the well-studied threshold algorithm (TA) [15,19,31]. Given an attribute list in each node, BPA accesses

Table 2
An example of a database with three nodes.

$Node_1$	$Node_2$	$Node_3$
(o_1, 12)	(o_{10}, 9)	(o_7, 11)
(o_2, 10)	(o_1, 8)	(o_2, 8)
(o_3, 8)	(o_3, 7)	(o_{10}, 6)
(o_4, 7)	(o_5, 6)	(o_3, 5)
(o_5, 5)	(o_7, 5)	(o_8, 5)
(o_6, 4)	(o_4, 4)	(o_1, 4)
(o_7, 3)	(o_6, 3)	(o_5, 3)
(o_8, 2)	(o_9, 2)	(o_9, 2)
(o_9, 1)	(o_2, 1)	(o_4, 1)
(o_{10}, 1)	(o_8, 1)	(o_6, 1)

those lists in parallel, one position at a time. At each iteration, the node reports to the coordinator the largest value in its attribute list which has not been reported yet. The coordinator calculates the scoring function $f()$ over these reported values. This score is referred to as a *threshold* (τ). In addition, the coordinator retrieves the objects for each accessed attribute and calculates each object's score. The k retrieved objects with the top scores are kept in a top-k list. Note that, since f is a monotonic function, τ bounds the score of unreported objects. Therefore, when the coordinator has been reported k objects (the top-k list) whose score is higher than τ, the coordinator can stop. While BPA is known to minimize the number of accessed objects in each node, this threshold-based technique requires many iterations between the coordinator and the nodes. This increases the overall execution times as well as the total number of objects reported to the coordinator, i.e., the communication bandwidth. As an example of how BPA proceeds, we study the data in Table 2. Each node holds one attribute for each object o_j. Assume that the scoring function $f()$ is the sum of the object's values over the nodes, and the query is for the top-2 scoring objects. In the first iteration, the coordinator looks at the first top objects of all nodes, which are o_1, o_{10}, and o_7. Then it looks them up over all nodes and computes their global score: $f(o_1) = 24, f(o_{10}) = 16, f(o_7) = 19$. The threshold value after one iteration is the sum of last reported values, i.e., $\tau = 12 + 9 + 11 = 32$. Since τ is larger than the current top-2 scored objects, the coordinator must continue to a second iteration, in which the reported objects are o_2 and o_3. Note that $node_2$ skips o_1, since the coordinator has already completed the attributes for o_1 from all nodes during the first iteration. Then the coordinator computes their global values $f(o_2) = 19$, $f(o_3) = 20$, and update the threshold value, $\tau = 10 + 7 + 8 = 25$. In the next iteration, $node_1$ returns o_4, $node_2$ returns o_5, and $node_3$ returns o_8. The coordinator computes their global score. The algorithm finally stops after this iteration since there are two objects o_1, o_3 whose global score ($f(o_1) = 24, f(o_3) = 20$) exceeds the current threshold, $\tau = 7 + 6 + 5 = 18$.

In order to reduce the number of iterations in distributed setups, a three-phase algorithm, (TPUT) [8], was developed. In the first phase, an estimated value of the kth object (τ) is determined. In the second, the coordinator computes a local threshold for each node, and the nodes report the objects which are locally above this threshold. We refer to this set of objects as the candidate list. Objects which are not reported are guaranteed not to be among the top-k. In the third phase, the coordinator sends the candidate list to all nodes and requests the local information of these objects, which has not been reported in previous phases. The advantage of this approach is that each phase consists of a single round-trip interaction between the nodes and the coordinator, which reduces the communications cost. While TPUT efficiently reduces the number of iterations, in case of d-dimensional vectorial aggregation queries it can become inefficient, as discussed later in this section. As an example of TPUT, we perform a top-2 query using the same scoring function $f()$ and dataset (Table 2) as the BPA algorithm. At the first phase, the coordinator requests all nodes to return their local top-2 objects ($\{o_1, o_2, o_{10}, o_7\}$), and computes their partial sum $\{P(o_1) = 20, P(o_2) = 18, P(o_{10} = 9), P(o_7) = 11\}$. Partial sum is an estimated score of each object computed by the coordinator, using only the reported set of attributes. The coordinator chooses the top-2 partial sums values, and takes the second one as a threshold, $\tau_1 = 18$. Then it sets a local threshold $T = \frac{\tau_1}{n} = \frac{18}{3} = 6$. During the second phase the nodes report the objects above this threshold: $node_1$ reports o_1, o_2, o_3 and o_4, $node_2$ reports o_{10}, o_1, o_3 and o_5, and $node_3$ reports o_7, o_2 and o_{10}. The coordinator updates the partial sum of each reported object, and sets $\tau_2 = \tau_1 = 18$ (since the top two partial sums are still o_1 and o_2). The coordinator also calculates the upper bound score of each object, and removes o_4 and o_5, whose upper bound score is below τ_2. After collecting all local vectors for each of the remaining objects, the coordinator determines o_1 and o_3 as the top-2 scored objects.

A three-phase approach was also used in the KLEE algorithm [29]. KLEE maintains a histogram of the data in each node. Upon request, part of the histogram is sent to the coordinator using Bloom filter compression. This enables returning an approximate top-k results with low communication cost. Briefly, the KLEE three phases follow like this. During the first phase the coordinator requests a fraction of the histogram from each node. Based on this information the coordinator computes an approximate top-k list and identifies the score of the kth object in this list as the top-k score. In the second phase, the coordinator uses the histogram data to create a candidate list of objects whose score may exceed the top-k score, and requests from each node an approximate value for each of these objects. In the third phase, the coordinator requests that the nodes complete the data for the objects whose approximate value exceed the top-k score.

In both the TPUT and KLEE algorithms, each node reports a single attribute to the coordinator, and the aggregation is performed over these attributes. In case each node holds a d-dimensional vectors, and the scoring function $f()$ is applied on the aggregation of these vectors, TPUT and KLEE view these attributes as different nodes, i.e., as a dn-dimensional function. For example, given 2 nodes and 3-dimensional vectors in each node i ($\vec{o}_{j,i} = A_{j,i}, B_{j,i}, C_{j,i}$), top-$k$ vectorial aggregation query over these vectors with the scoring function $f(o_j) = f\left(\frac{\vec{o}_{j,1}+\vec{o}_{j,2}}{2}\right)$ will be redefined as $f(o_j) = f\left(A_{j,1}, A_{j,2}, B_{j,1}, B_{j,2}, C_{j,1}, C_{j,2}\right)$, where the scoring function includes the computation of the average. In case of high-dimensional vectors and a large number of nodes, the definition of the local thresholds in each node becomes very difficult and inefficient, and incur high computational, storage accesses and especially communication costs.

The BPA, TPUT and KLEE algorithms reduce the communication bandwidth between the nodes and the coordinator. We measure the communication bandwidth by counting the total number of objects sent from and to the coordinator. That is, each object the nodes chose to report to the coordinator is counted as a single transmission (the beginning of each BPA iteration, and TPUT and KLEE first two phases), while each object collected as a result of a coordinator request is counted as two transmissions, one in which the coordinator sends each node a list of requested objects, and the second in which the nodes reply with the local vectors for these objects.

In this paper, we propose an algorithm to reduce the communication cost by defining efficient local thresholds over d-dimensional local statistics vectors in each node. In some cases, as shown in the experimental section, our algorithm can reduce communication cost by up to four orders of magnitude relative to BPA, TPUT and KLEE.

2.2. Our approach

To achieve low communication cost and latency, our distributed top-k algorithm is designed with four phases, each consisting of a single round-trip interaction between the nodes and a coordinator:

(1) Phase I—the coordinator determines a lower bound, τ_1, on the score of the top-k objects.
(2) Phase II—the coordinator collects additional information from the nodes and determines an improved lower bound, τ_2.
(3) Phase III—each node uses the bound τ_2 to prune out objects that are guaranteed not to be in the top-k scoring set, and sends the remaining objects to the coordinator.
(4) Phase IV—the coordinator collects additional information from the nodes to determine the top-k objects among those it received in the previous stages.

The suggested algorithm draws its efficiency over the previous algorithms through its ability to compute the value of an object over vectorial aggregation, reducing the number of attributes participating in the computation of the scoring function. This vectorial approach enables defining tight and efficient local thresholds in each node. In the next sections we describe these phases and the algorithm in detail.

3. Top-k vectorial aggregation query algorithm

The algorithm consists of four phases, each taking one round-trip to complete. In this section we assume the data is static and that there is a single coordinator. In Section 6 we present a version of the algorithm which handles data updates and scalability problems.

3.1. Phase I—determining the lower bound

The goal of the first phase is to determine a lower bound on the score of the top-k objects. We first calculate a lower bound on the scores of certain k objects (not necessarily the top scoring ones). The minimum among these will be the lower bound we seek because at least k objects will have a score equal to or exceeding this value. The first phase proceeds with each node determining the k highest scoring objects according to the local statistics vectors held for them. This is done by computing the function $f()$ at the vectors $x_{j,i}$ (note that the local vectors are of the same dimension as the global ones, hence the scoring function f can be applied to them). Next, each node sends the coordinator the local statistics vector of the k selected objects.

Once the coordinator received the objects from the nodes, it compiles them to a single list. The coordinator records the local statistics vector received for each object and which nodes reported them. If several nodes reported a certain object, the coordinator records the average of the local statistics vectors received for it. The vector recorded for each object in the list is referred to as its *partial statistics vector*. The list of objects maintained by the coordinator is called the *partial statistics list*. The partial statistics vector held for the object o_j is denoted by $\vec{p}_j$. The multiplicity of an object in the partial statistics list is defined as the number of nodes that reported their local statistics vectors for it, and is denoted by m_j. We say the coordinator has *complete statistics* for a certain object if all the nodes have reported their local statistics vectors for it, i.e., the multiplicity of the object is n.

Given an object for which the coordinator holds a partial statistics vector $\vec{p}_j$ and whose multiplicity is m_j, note that $m_j\vec{p}_j$ is the sum of the local statistics vectors received for the object, and therefore, according to Eq. (1), $\vec{x}_j \succ \frac{m_j\vec{p}_j}{n}$. Since $f()$ is monotonic, it follows that $f(\vec{x}_j) \geq f\left(\frac{m_j\vec{p}_j}{n}\right)$. Therefore, $f\left(\frac{m_j\vec{p}_j}{n}\right)$ is a lower bound on the score of the object. The coordinator determines this lower bound for all the objects in the partial statistics list and selects the k objects whose lower bound is the highest. These objects are referred to as the *initial candidate set*. The objects in the initial candidate set are sorted according to their lower bounds. The lower bound of the kth object in the list is denoted by τ_1. As explained above, τ_1 is a lower bound on the scores of the top-k scoring objects.

3.2. Phase II—improving the lower bound

In the second phase we improve τ_1, the lower bound calculated in the first phase, by collecting all the local statistics vectors for the objects in the initial candidate set. At the end of this phase the coordinator can determine the exact scores of these objects; it then sorts them according to their score. The score of the kth object is denoted by τ_2. Note that $\tau_2 \geq \tau_1$.

While the second phase is not required for the correctness of the algorithm (it can be skipped, and τ_2 can be replaced by τ_1), experimental results show that this phase notably improves the lower bound, leading to a dramatic reduction in the communication cost of subsequent phases.

3.3. Phase III—local elimination of objects

The goal of the third phase is to determine a set of objects guaranteed to include the top-k scoring objects. Each node locally detects objects whose global score might exceed τ_2 and reports them to the coordinator. Since τ_2 is a lower bound on the scores of the top-k objects, we can be certain that objects which have not been reported by any node are not among the top-k objects.

As demonstrated in the Introduction, distributively determining that the score of an object does not exceed a given threshold may be difficult, since combining local statistics vectors whose local score is smaller than τ_2 can yield a global statistics vector whose score is above τ_2. We address this problem using *tentative upper bounds*.

Tentative upper bounds are scalar values determined locally for an object by each node. The tentative upper bound determined by the node p_i for the object o_j is denoted $u_{j,i}$. Let U_j be the set of tentative upper bounds determined for o_j by the various nodes, i.e., $U_j = \max_i\{u_{j,1}, u_{j,2}, \ldots, u_{j,n}\}$. Then, the following theorem holds:

Theorem 1. $f(\vec{x}_j) \leq U_j$.

In Section 4.1, we describe a geometric method for ensuring that the global score of an object will not exceed one of the tentative upper bounds determined for it, and prove Theorem 1.

The third phase begins with the coordinator sending the value of τ_2 to all the nodes. Each node compiles a list of all the objects whose tentative upper bound exceeds τ_2. This list is referred to as the *local candidate set*. The nodes send the local statistics vectors of the objects in their local candidate sets to the coordinator. Tentative upper bounds guarantee that unreported objects cannot be among the top-k objects, and therefore can be pruned without communication. This is the most important property of these bounds. In Section 4.2 we show that constructing the local candidate set does not require the nodes to iterate over all the objects, and in Section 4.3 we show how to efficiently reduce the computational cost. After receiving the local candidate sets from all the nodes, the coordinator adds the local statistics vectors from the candidate sets to the partial statistics list.

If several nodes have sent the coordinator their local statistics vectors for o_j, it can determine a tentative upper bound of o_j that will replace those determined by the nodes. Say the nodes $p_1, p_2, \ldots, p_l$ have sent the coordinator their local statistics vector for o_j. The coordinator determines a tentative upper bound for o_j, which is denoted by $u_{j,0}$. In this case, the value of

$U_j = \max_i\{u_{j,0}, u_{j,l+1}, u_{j,l+2}, \ldots, u_{j,n}\}$ satisfies Theorem 1. The coordinator can now locally prune additional candidates from the partial statistics list. The coordinator might now hold complete statistics for new objects (in addition to the k objects it collected complete statistics for in the second phase). This enables it to determine a better lower bound on the score of the top-k objects. It examines the list of objects for which it has complete statistics, and sets τ_3 to be the score of the kth scoring object. Note that $\tau_3 \geq \tau_2$. Finally, the coordinator determines tentative upper bounds for the objects in the partial statistics list and removes all those objects whose tentative bound is below τ_3.

3.4. Phase IV—determining the top-k scoring objects

In the fourth phase, the coordinator determines the top-k scoring objects among those in the partial statistics list by requesting the unknown local statistics vectors for all the objects in the list. This can be accomplished in a single round-trip interaction. Once the coordinator has determined the global statistics vectors for all the objects, it can calculate their final score. It then selects the k top scoring objects.

Note that the fourth phase can be very costly in terms of communication. In Section 5 we show that many objects can be eliminated at a fraction of the cost of collecting all their local statistics vectors by slightly increasing the latency of this phase.

The algorithm is summarized in Algorithm 1.

Algorithm 1 Top-k Vectorial Aggregation Query

1. Phase1 :
   For each node i:
   (a) Let $list_i$ be the top k objects
   (b) Report $list_i$ to coordinator
   The coordinator:
   (a) $list = \bigcup list_i$.
   (b) For each $o_j \in list$ compute its partial sum vector($\vec{p}_j$).
   (c) Compute the partial sum score ($f(\vec{p}_j)$) of each object, and let $list_k$ be the k top scored objects.
   (d) Let o_r be the object in the kth position. Let $\tau_1 = f(\vec{p}_r)$.
   (e) Report $list$ to all nodes
2. Phase2 :
   For each node i:
   (a) return $\vec{x}_{j,i}$ for each $o_j \in list$
   The coordinator:
   (a) For each $o_j \in list$, compute $f(o_j)$.
   (b) Let τ_2 be the kth top scoring object
   (c) Report τ_2 to all nodes
3. Phase3 :
   For each node i:
   (a) Let $lcl_i = \{o_j | u_{j,i} \geq \tau_2\}$
   (b) Report lcl_i to coordinator
   The coordinator:
   (a) Let $gcs = \bigcup lcl_i$
   (b) Report gcs to nodes
4. Phase4 :
   For each node i:
   (a) For each $o_j \in gcs$, report $\vec{x}_{i,j}$ to coordinator
   The coordinator:
   (a) For each $o_j \in gcs$, compute $f(o_j)$
   (b) return the k top scoring objects

4. Determining tentative upper bounds

4.1. Definition of tentative upper bounds

In this section we describe how the nodes determine tentative upper bounds. Recall that tentative upper bounds are local constraints in each node which guarantee that for every object there exists at least one tentative upper bound whose score exceeds the object's global score. Therefore, if the object's tentative upper bounds in all nodes are below a given threshold, the object's global score is guaranteed to be below this threshold and it can be discarded.

Recall that $\vec{x}_{j,1}, \vec{x}_{j,2}, \ldots, \vec{x}_{j,n}$ are the local statistics vectors held for the object o_j by the various nodes, and that $\vec{x}_j$, the global statistics vector for o_j, is their average:

$$\vec{x}_j = \sum_{i=1}^{n} \frac{\vec{x}_{j,i}}{n}. \tag{2}$$

A geometric interpretation of Eq. (2) is that an object's global statistics vector belongs to the convex hull of its local statistics vectors. Let $\mathrm{Conv}(\vec{x}_{j,1}, \vec{x}_{j,2}, \ldots, \vec{x}_{j,n})$ denote the convex hull of the local statistics vectors held by the nodes for o_j. Then $\vec{x}_j \in \mathrm{Conv}(\vec{x}_{j,1}, \vec{x}_{j,2}, \ldots, \vec{x}_{j,n})$.

Given an agreed common *reference vector*, $\vec{x}_{\mathrm{ref}}$, each node constructs a sphere centered at the midpoint between $\vec{x}_{j,i}$ and the reference vector, with a radius equal to half the distance between them. This sphere is referred to as the *bounding sphere* and is denoted by $B(\vec{x}_{\mathrm{ref}}, \vec{x}_{j,i})$:

$$B(\vec{x}_{\mathrm{ref}}, \vec{x}_{j,i}) = \left\{ \vec{z} \;\middle|\; \left\| \vec{z} - \frac{\vec{x}_{\mathrm{ref}} + \vec{x}_{j,i}}{2} \right\| \leq \left\| \frac{\vec{x}_{\mathrm{ref}} - \vec{x}_{j,i}}{2} \right\| \right\}.$$

According to Theorem 2, proved in [40], the union of the bounding spheres created by the nodes for a certain object contains the convex hull of its local statistics vectors.

Theorem 2. *Let $\vec{x}_{\mathrm{ref}}, \vec{x}_{j,1}, \vec{x}_{j,2}, \ldots, \vec{x}_{j,n} \in \mathbb{R}^d$ be a set of vectors in $\mathbb{R}^d$. Let $\mathrm{Conv}(\vec{x}_{\mathrm{ref}}, \vec{x}_{j,1}, \vec{x}_{j,2}, \ldots, \vec{x}_{j,n})$ be the convex hull of $\vec{x}_{\mathrm{ref}}, \vec{x}_{j,1}, \vec{x}_{j,2}, \ldots, \vec{x}_{j,n}$. Let $B(\vec{x}_{\mathrm{ref}}, \vec{x}_{j,i})$ be a sphere centered at $\frac{\vec{x}_{\mathrm{ref}} + \vec{x}_{j,i}}{2}$ and with a radius of $\left\| \frac{\vec{x}_{\mathrm{ref}} - \vec{x}_{j,i}}{2} \right\|$. That is, $B(\vec{x}_{\mathrm{ref}}, \vec{x}_{j,i}) = \left\{ \vec{z} \;\middle|\; \left\| \vec{z} - \frac{\vec{x}_{\mathrm{ref}} + \vec{x}_{j,i}}{2} \right\| \leq \left\| \frac{\vec{x}_{\mathrm{ref}} - \vec{x}_{j,i}}{2} \right\| \right\}$.*
Then $\mathrm{Conv}(\vec{x}_{\mathrm{ref}}, \vec{x}_{j,1}, \vec{x}_{j,2}, \ldots, \vec{x}_{j,n}) \subset \bigcup_{i=1}^{n} B(\vec{x}_{\mathrm{ref}}, \vec{x}_{j,i})$.

Fig. 2 illustrates the use of the geometric bounding technique. The four local statistics vectors are depicted together with the reference vector. In addition, the figure depicts the bounding spheres constructed by each node. One can see that the union of the spheres contains the convex hull of the local statistics vectors.

Corollary 3. *The global vector $\vec{x}_j$ belongs to the union of the bounding spheres, $\vec{x}_j \in \bigcup_{i=1}^{n} B(\vec{x}_{\mathrm{ref}}, \vec{x}_{j,i})$.*

Let $u_{j,i}$ be the maximum score received on the vectors by node i for object o_j's bounding sphere, i.e., $u_{j,i} = \max_{\vec{v} \in B(\vec{x}_{\mathrm{ref}}, \vec{x}_{j,i})}(f(\vec{v}))$. We denote by $u_{j,i}$ the *tentative upper bound* of object o_j in node i.

According to Corollary 3 and the definition of $u_{j,i}$, the tentative upper bound determined by the node p_i is guaranteed to be equal to or exceed the score of the object o_j, proving Theorem 1.

The $\vec{x}_{\mathrm{ref}}$ we use here is defined as follows. Given the set of all local statistics vectors of all objects, let B be their minimal axis-aligned bounding box. We denote the corner of B with the minimal value in each component as $\vec{x}_{\mathrm{ref}}$ ($\vec{x}_{\mathrm{ref}}$ might not correspond to an actual data vector). Clearly $\vec{x}_{\mathrm{ref}}$ is dominated by all the local statistics vectors, i.e., $\forall \vec{x}_{j,i} \succ \vec{x}_{\mathrm{ref}}$. We assume that each node maintains its local data structure to allow the immediate retrieval of the object with minimal value for that attribute. Then the coordinator can easily compute $\vec{x}_{\mathrm{ref}}$ by collecting these objects from each node in a single communication phase, and report it to the nodes.

Real-world data is often not distributed identically along the various axes. In such cases, we may obtain tighter tentative upper bounds by using ellipsoidal bounding regions instead of

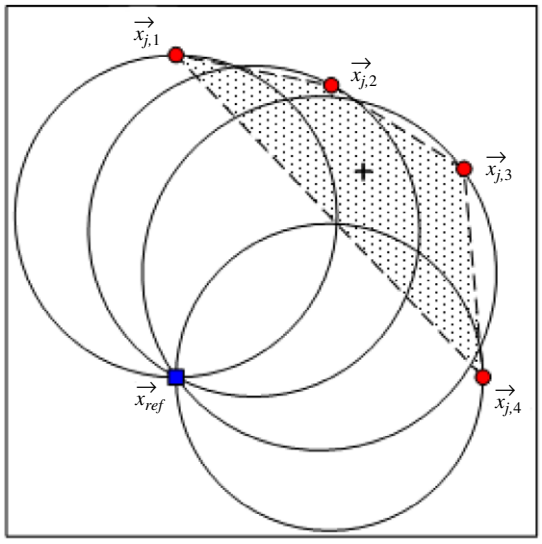


Fig. 2. Using spherical bounds. Four local statistics vectors are depicted (red circles). The reference vector (blue square), and the bounding spheres determined by the nodes, are also depicted. Note that the convex hull of the local vectors is contained in the union of the spheres.

the spherical bounding regions described in the previous section. Fig. 3(a) depicts the local statistics vectors held by four nodes for a certain object. Note that the variance of the data along the x-axis is considerably higher than the variance of the data along the y-axis. Thus, before determining the spherical bounding regions, we would like to scale the data so that its variance along all the dimensions is identical. Fig. 3(b) depicts the data after the scaling process. Next we determine the spherical bounding regions (Fig. 3(c)), and finally, we reverse the scaling process (Fig. 3(d)). One can see that using ellipsoidal bounds yields smaller bounding regions, and therefore they are likely to yield tighter tentative upper bounds.

Employing ellipsoidal bounding regions requires the nodes to agree on scaling parameters, which can be reused for answering multiple queries. In order to determine common scaling parameters, each node determines the average and variance of the data along each axis. The coordinator collects these statistics from the nodes, calculates the global variance of the data along each axis, and sends these global scaling parameters to the nodes. The global scaling parameters will be used by the nodes to create bounding ellipsoids in subsequent top-k queries.

Experimental results (see Section 7) demonstrate that communication costs for top-10 queries are up to three orders of magnitude lower when ellipsoidal bounding regions are used, and well over an order of magnitude lower for top-100 queries.

4.1.1. Determining tentative upper bounds at the coordinator

Up to this point we defined the local tentative upper bound on the score of each object computed in each node. In this section we describe how the coordinator can determine a tentative upper bound by using the local tentative upper bounds reported by various nodes. Moreover, we prove that the tentative upper bound determined by the coordinator does not exceed the maximum among the tentative upper bounds determined in each of the nodes.

Assume without loss of generality that the nodes $p_1, p_2, \ldots, p_l$ ($l \leq n$) have sent the coordinator their local statistics vector for o_j. Recall that the coordinator records the average of the partial

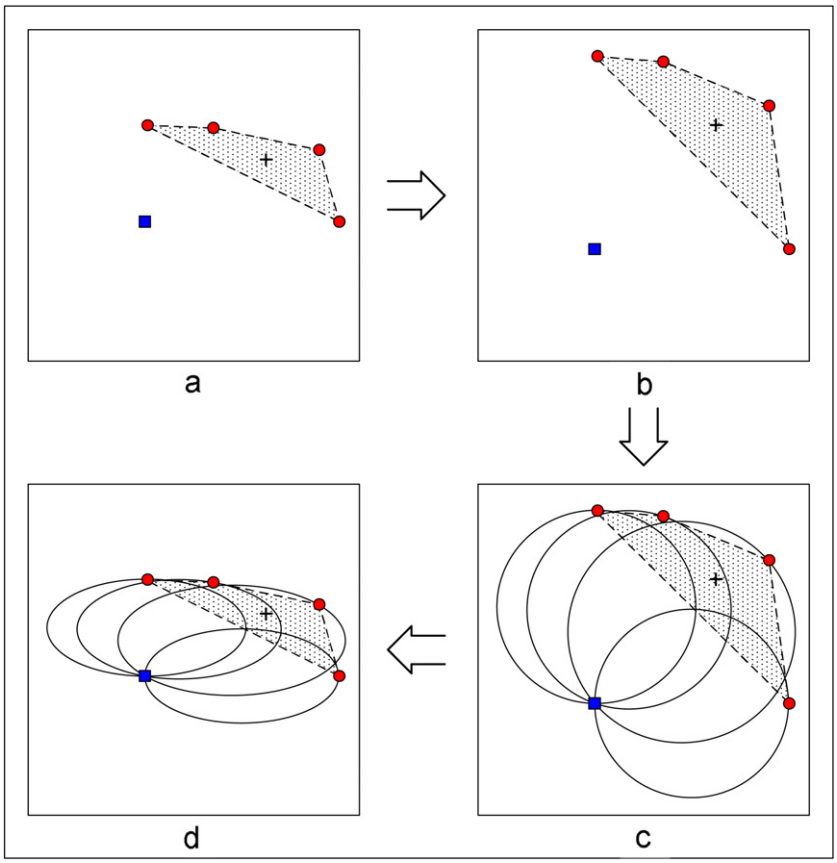


Fig. 3. Employing ellipsoidal bounding regions. Data are usually not distributed identically along the various dimensions (a). The scaling process (b, c, d) yields ellipsoidal bounding regions (d). Note that the reference vector (blue square) is constrained to lie at the origin, in order to satisfy the aforementioned domination constraint.

statistics vectors it received for o_j in the partial statistics list. The partial statistics vector maintained for o_j is denoted by $\vec{p}_j$. The global statistics vector for o_j can be written as follows:

$$\vec{x}_j = \sum_{i=1}^{n} \frac{\vec{x}_{j,i}}{n} = \frac{l}{n}\vec{p}_j + \sum_{i=l+1}^{n} \frac{\vec{x}_{j,i}}{n}. \quad (3)$$

A geometric interpretation of Eq. (3) is that the global statistics vector belongs to the convex hull of $\vec{p}_j, \vec{x}_{j,l+1}, \vec{x}_{j,l+2}, \ldots, \vec{x}_{j,n}$. Let the coordinator construct the sphere $B(\vec{x}_{\text{ref}}, \vec{p}_j)$ and determine the tentative upper bound $u_{j,0}$, which is the maximum score received by the vectors that belong to the sphere $B(\vec{x}_{\text{ref}}, \vec{p}_j)$. According to Theorem 2, the convex hull is contained in the union of the spheres $B(\vec{x}_{\text{ref}}, \vec{x}_{j,l+1})$, $B(\vec{x}_{\text{ref}}, \vec{x}_{j,l+2}), \ldots, B(\vec{x}_{\text{ref}}, \vec{x}_{j,n})$ and $B(\vec{x}_{\text{ref}}, \vec{p}_j)$. Consequently, the score of the object o_j does not exceed the maximum among $u_{j,l+1}, u_{j,l+2}, \ldots, u_{j,n}$ and $u_{j,0}$.

Next, we show that the tentative upper bound determined by the coordinator does not exceed the maximum among the tentative upper bounds determined by the nodes that sent it their local vectors, i.e., $u_{j,0} \leq \max(u_{j,1}, u_{j,2}, \ldots, u_{j,l})$. This property is based on the following theorem:

Theorem 4. *Let $\vec{x}_{j,1}, \vec{x}_{j,2}, \ldots, \vec{x}_{j,l}$ be d-dimensional vectors. Let $\vec{p}_j$ be the average of the partial statistics vectors sent by nodes $1, 2, \ldots, l$. Then $u_{j,0} \leq \max(u_{j,1}, u_{j,2}, \ldots, u_{j,l})$.*

Proof. Since $\vec{p}_j$ is a weighted sum of the vectors $\vec{x}_{j,1}, \vec{x}_{j,2}, \ldots, \vec{x}_{j,l}$, then according to the geometric interpretation, $\vec{p}_j \in \text{Conv}(\vec{x}_{j,1}, \vec{x}_{j,2}, \ldots, x_{j,l})$. According to Corollary 3, $\vec{p}_j \in \bigcup_{i=1}^{l} B(\vec{x}_{\text{ref}}, \vec{x}_{j,i})$. Therefore, $u_{j,0} \leq \max(u_{j,1}, u_{j,2}, \ldots, u_{j,l})$. □

Consequently, the tentative upper bound determined by the coordinator is guaranteed not to exceed the maximum among the tentative upper bounds determined by these nodes.

4.2. Minimizing local accesses

The local candidate set, described in the previous section, contains all the objects whose tentative upper bound exceeds a given threshold τ_2. However, calculating the tentative upper bound value for every object in each node for every query incurs high local storage access cost. In this section we show how to compute the local candidate set by computing its value for only a fraction of the objects in each node, by using a progressive detection of skyline objects.

Given a set of objects, their *skyline* contains the objects in the set whose local statistics vectors are not dominated by any local statistics vector. Recall that a vector $\vec{x}$ dominates a vector $\vec{y}$ ($\vec{x} \succ \vec{y}$) if no component of $\vec{x}$ is smaller than the corresponding component of $\vec{y}$. Given a set of objects in a node p_i, we can therefore bound their tentative upper bound value by bounding the tentative upper bound of the skyline objects.

Lemma 5. *Let $\vec{x}_{a,i}$ and $\vec{x}_{b,i}$ be local statistics vectors for two objects at the node p_i. Furthermore, let $\vec{x}_{a,i}$ dominate $\vec{x}_{b,i}$. Then $u_{a,i} \geq u_{b,i}$. (The tentative upper bound value of the first object is greater than the tentative upper bound value of the second object.)*

Proof. We start by observing the following properties of the domination relationship: given three vectors $\vec{v}_1, \vec{v}_2$, and $\vec{v}_3$, where $\vec{v}_1$ dominates $\vec{v}_2$, and $\vec{v}_2$ dominates $\vec{v}_3$:

1. The domination relationship is transitive, i.e., $\vec{v}_1$ dominates $\vec{v}_3$.
2. For *any* monotonic function f, $f(\vec{v}_1) \geq f(\vec{v}_2) \geq f(\vec{v}_3)$.
3. The translation operation preserves the domination relationship, i.e., given a constant vector $\vec{y}$, $\vec{v}_1 + \vec{y}$ dominates $\vec{v}_2 + \vec{y}$.
4. Multiplication by a positive scalar preserves the domination relationship, i.e., given $\alpha > 0$, $\alpha\vec{v}_1$ dominates $\alpha\vec{v}_2$.
5. The distance between $\vec{v}_1$ and $\vec{v}_3$ is greater than the distance between $\vec{v}_3$ and $\vec{v}_2$, i.e., $\|\vec{v}_1 - \vec{v}_3\| > \|\vec{v}_2 - \vec{v}_3\|$.

Given two objects o_a and o_b, where o_a locally dominates o_b at the node p_i (i.e., $\vec{x}_{a,i}$ dominates $\vec{x}_{b,i}$), we make several observations. Recall our requirement that $\vec{x}_{\text{ref}}$ be dominated by all the local statistics vectors. Thus, $\vec{x}_{b,i}$ dominates $\vec{x}_{\text{ref}}$. Next, we examine $B(\vec{x}_{\text{ref}}, \vec{x}_{a,i})$ and $B(\vec{x}_{\text{ref}}, \vec{x}_{b,i})$, the bounding spheres for o_a and o_b. The centers of these spheres are $\frac{\vec{x}_{\text{ref}}+\vec{x}_{a,i}}{2}$ and $\frac{\vec{x}_{\text{ref}}+\vec{x}_{b,i}}{2}$ respectively. We denote these centers by $\vec{c}_{a,i}$ and $\vec{c}_{b,i}$. According to (3) and (4) above, $\vec{c}_{a,i}$ dominates $\vec{c}_{b,i}$. In addition, the radius of the bounding sphere created for o_a is greater than the radius of the bounding sphere created for o_b (follows from (5) above).

Recall that $u_{j,i}$, the tentative upper bound for an object o_j, is the maximum score received for the vectors in the object's bounding sphere, i.e.,

$$u_{j,i} = \max_{\vec{v} \in B(\vec{x}_{\text{ref}}, \vec{x}_{j,i})} (f(\vec{v})).$$

We claim that for any monotonic scoring function f, $u_{a,i} \geq u_{b,i}$. Let $\vec{z} \in B(\vec{x}_{\text{ref}}, \vec{x}_{b,i})$ be a vector such that $f(\vec{z}) = u_{b,i}$, i.e., $\vec{z}$ is the vector in the bounding sphere defined for o_b that maximizes the scoring function f. Now we show how to match $\vec{z}$ with a vector $\vec{z'} \in B(\vec{x}_{\text{ref}}, \vec{x}_{a,i})$, such that $\vec{z'}$ dominates $\vec{z}$. Since $\vec{z'}$ belongs to $B(\vec{x}_{\text{ref}}, \vec{x}_{a,i})$, its score does not exceed $u_{a,i}$, i.e.,

$$f(\vec{z'}) \leq u_{a,i}.$$

Since $\vec{z'}$ dominates $\vec{z}$, its score is equal to or higher than the score of $\vec{z}$, which is $u_{b,i}$, i.e.,

$$u_{b,i} \leq f(\vec{z'}).$$

It follows that $u_{a,i} \geq u_{b,i}$, as desired.

We match $\vec{z'}$ to $\vec{z}$ as follows. Let $\delta(\vec{z})$ be the offset of $\vec{z}$ from the center of the sphere, i.e.,

$$\delta(\vec{z}) = \vec{z} - \vec{c}_{b,i}.$$

Then

$$\vec{z'} = \vec{c}_{a,i} + \delta(\vec{z}).$$

Fig. 4 illustrates the constructs described above. It depicts the reference vector, $\vec{x}_{\text{ref}}$ (blue square), the local statistics vector $\vec{x}_{a,i}$ for the object o_a (green circle), and the vector $\vec{x}_{b,i}$ for o_b (red circle). Note that $\vec{x}_{a,i}$ dominates $\vec{x}_{b,i}$. The bounding spheres for each object are depicted. The centers of the bounding spheres are depicted by the green and red diamonds. The figure depicts a vector $\vec{z}$ that belongs to $B(\vec{x}_{\text{ref}}, \vec{x}_{b,i})$, and the corresponding vector $\vec{z'}$ in $B(\vec{x}_{\text{ref}}, \vec{x}_{a,i})$.

Note that since the radius of the bounding sphere for o_a is larger than the radius of the bounding sphere for o_b, if $\vec{z}$ belongs to $B(\vec{x}_{\text{ref}}, \vec{x}_{b,i})$, then $\vec{z'}$ belongs to $B(\vec{x}_{\text{ref}}, \vec{x}_{a,i})$, as desired. In addition, note that we can express $\vec{z}$ and $\vec{z'}$ as follows:

$$\vec{z} = \vec{c}_{b,i} + \delta(\vec{z})$$
$$\vec{z'} = \vec{c}_{a,i} + \delta(\vec{z}).$$

Since $\vec{c}_{a,i}$ dominates $\vec{c}_{b,i}$, according to (3) above, $\vec{z'}$ dominates $\vec{z}$, as desired. As described above, it follows that $u_{a,i} \geq u_{b,i}$. □

Theorem 6. *Let S_i be the set of skyline objects computed over a set of objects in node p_i and a given threshold τ_2. If for every object o_j in S_i, $u_{j,i} < \tau_2$, then the tentative upper bound value for every object in node p_i is below τ_2.*

Proof. Let $\vec{x}_{j,i}$ be the local statistics vector of object o_j in node p_i. If $o_j \in S_i$, then by definition $u_{j,i} < \tau_2$. If $o_j \notin S$, then according

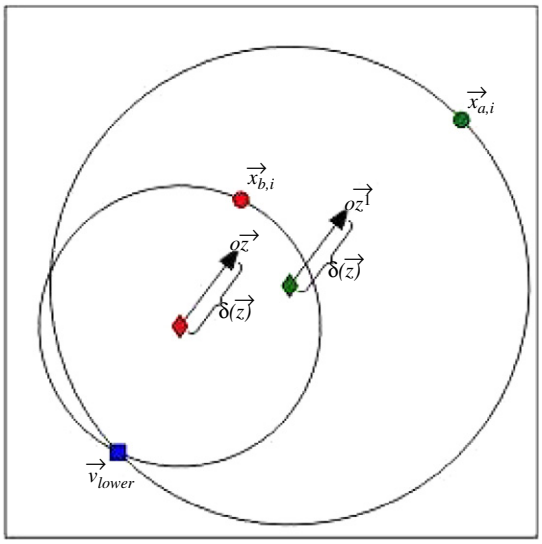


Fig. 4. Two local vectors $\vec{x}_{a,i}$ and $\vec{x}_{b,i}$ are depicted, such that $\vec{x}_{a,i}$ dominates $\vec{x}_{b,i}$. The tentative upper bound defined by $\vec{x}_{a,i}$ is guaranteed not to be smaller than that defined by $\vec{x}_{b,i}$.

to the skyline definition, there exists an object o_k, $o_k \in S$, such that $\vec{x}_{k,i} \succ \vec{x}_{j,i}$. Since $u_{j,i} \leq u_{k,i}$ (Lemma 5) and $u_{k,i} < \tau_2$, then $u_{j,i} < \tau_2$. □

Using Theorem 6, we present a progressive algorithm (Algorithm 2) for constructing the local candidate set while minimizing the number of local accesses. Let O be a set of objects in node p_i. Upon initialization, this set contains all the objects in p_i. At each iteration, node p_i computes the skyline objects set (S) of O, and computes the tentative upper bound value of each object in this set. If the tentative upper bound value of an object $o_j \in S$ is above τ_2, then o_j is removed from O and added to the local candidate set. Otherwise, this object is kept in O. The iteration process terminates when the tentative upper bound value of all objects in S is below τ_2. According to Theorem 6, since the tentative upper bound value of each object in S is below τ_2, then the tentative upper bound values of all current objects in O are below τ_2 as well. In addition, the local candidate set is now guaranteed to consist of the objects whose tentative upper bound exceeds τ_2.

Algorithm 2 Constructing local candidate list

1. Let O be the set of all objects in node p_i, and τ_2 a given threshold.
2. Compute the skyline objects set (S) of O.
3. For each object $o_j \in S$, compute the local upper tentative bound $u_{j,i}$. If $u_{j,i} \geq \tau_2$, then remove o_j from O and add it to the local candidate set.
4. If the set O has been changes - return to 2.
5. Return the objects in the local candidate set.

Various algorithms were proposed for efficiently computing the skyline. These algorithms can be classified into two categories: those which do not use pre-processing [5,17,14] (and therefore have to make at least one pass over the database) and those which use pre-processing [24,43,35] (such as indexing and sorting). Since in many database applications attribute values are already indexed in a tree structure (e.g., B-tree, R-tree), here we have chosen to apply the branch-and-bound technique for skyline computation (BBS), proposed by Papadias et al. in [35]. The BBS algorithm explores an R-tree using a best-first search paradigm, and has been shown to be optimal with respect to R-tree page accesses. Experiments, presented in Section 7, show that progressive detection of skyline objects using BBS reduces the number of local accesses by two orders of magnitude in comparison to accessing all the objects in each node.

4.3. Minimizing computational costs

The top-k vectorial aggregation algorithm (Algorithm 1) presented in the previous section requires that the tentative upper bound value be computed for every object. Finding the tentative upper bound value can be viewed as an optimization problem of the following form:

$$\max(f(\vec{x}_{j,i})),$$
$$\vec{x}_{j,i} \in D$$

where f is referred to as the *objective function*, and D is referred to as the *feasibility region* (the bounding sphere $B(\vec{x}_{\text{ref}}, \vec{x}_{j,i})$ in our case). The difficulty in solving optimization problems is that many points in the feasibility region might locally maximize the objective function. Consequently, local algorithms, such as gradient descent, may not be suitable. In this section we present a branch-and-bound method for determining the tentative upper bound value.

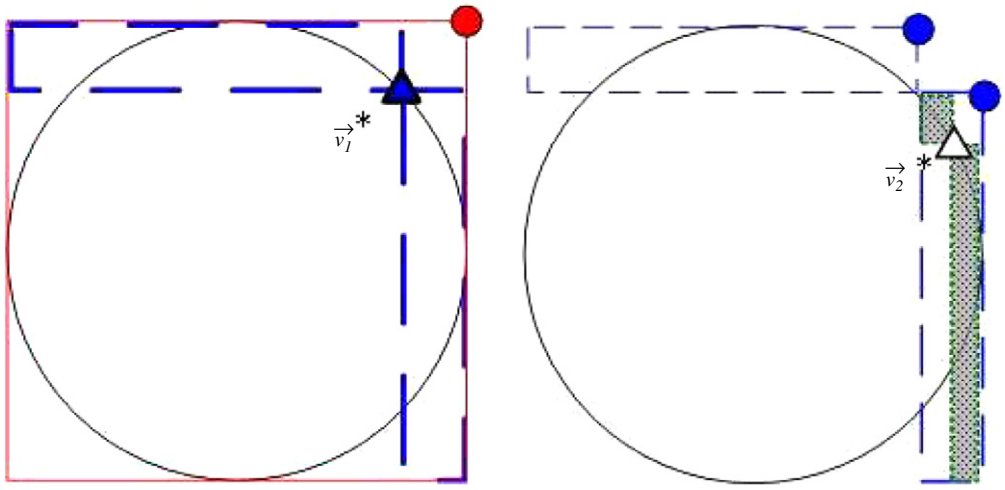
4.4. The branch-and-bound method

Given a threshold τ, our goal is to determine whether the tentative upper bound value of an object's local statistics vector $\vec{x}_{j,i}$ is above τ, by determining the maximum score received in the sphere $B(\vec{x}_{\text{ref}}, \vec{x}_{j,i})$. For many scoring functions, it may be difficult to directly determine the maximum score received in a sphere, but easy to determine the maximum score received in a box. For monotonic scoring functions, for example, the maximum score in a box is always obtained at the corner that has the maximum value in all coordinates.

Using a branch-and-bound method, we cover the sphere with a set of boxes, and compute the maximum scores received in them. If the maximum scores are below τ, then the tentative upper bound value of the object is also guaranteed to be below τ. If the maximum scores are not below τ, we can increase the number of boxes, creating a tighter cover of the sphere.

Given a sphere $B(\vec{x}_{\text{ref}}, \vec{x}_{j,i})$, a monotonic scoring function $f()$, and a threshold value τ, the branch-and-bound method (as described in Algorithm 3) initializes with the set of boxes containing a single box that tightly encloses the sphere. If the maximum value on the box is below τ, the algorithm terminates; otherwise it refines this set in the next iteration. Let B_i be the set of boxes maintained by the algorithm in the ith iteration. At the beginning of each iteration, the algorithm selects the box in which the maximum score is received. We denote this box by b_i. If b_i's maximum score is below the threshold, the algorithm terminates, having determined that the maximum value of this sphere is below τ. Otherwise, the algorithm determines the vector of the sphere that is closest to b_i's corner. This vector is denoted by $\vec{v}^*_i$. If $f(\vec{v}^*_i)$ is above τ, the algorithm terminates and reports this object. Otherwise, $\vec{v}^*_i$ is used to partition b_i into 2^d sub-boxes, as depicted in Fig. 5. The algorithm defines the set B_{i+1} to include all the boxes in B_i. It then proceeds to the next iteration, excluding b_i, and all the sub-boxes of b_i. After a finite number of iterations, if there is a box whose maximum score exceeds the given threshold, then the algorithm terminates and reports this object.

As the number of iterations increases, the maximum score received in the set of boxes is closer to the tentative upper bound value. Experiments with the branch-and-bound method over a real data set (AOL query logs) have shown that the average number of iterations needed to determine whether the tentative upper bound value of an object is above a threshold τ_2 is very low, the average number of iterations in 50,000 runs was 1.95 and the standard deviation 0.054.



(a) 1st Iteration. (b) 2nd Iteration.

Fig. 5. The branch-and-bound method. In the first iteration (a), the maximum score of the red (solid) box is above the threshold, and this box is partitioned to two blue (dashed line) sub-boxes. In the next iteration (b), the right blue box is selected and partitioned to the two grey (dotted) sub-boxes.

Algorithm 3 Checking if an object's tentative upper bound is above a given threshold

1. Given $\vec{x}_{j,i}$ - the local statistics vector of object o_j in node p_i; a reference vector $\vec{x}_{\text{ref}}$; a scoring function f; and a threshold τ.
2. Let $i = 0$ - An iteration counter.
3. Compute the sphere $B(\vec{x}_{\text{ref}}, \vec{x}_{j,i})$. Let B_i be the tightly enclosing box of this sphere.
4. While $i <$number of iterations:
   (a) Let $b_i \in B_i$ be the box with the maximum score according to f.
   (b) If b_i's score is below τ - the tentative upper bound of o_j is below the threshold and the algorithm terminates without reporting it.
   (c) Otherwise, choose the vector of the sphere that is closest to b_i's corner. Denote this vector by $\vec{v^*}_i$.
   (d) If $f(\vec{v^*}_i) \geq \tau$, report object o_j and terminate.
   (e) Otherwise, partition b_i into 2^d sub-boxes using $\vec{v^*}_i$.
   (f) Set B_{i+1} to include all the boxes in B_i and all the sub-boxes of b_i, excluding b_i.
   (g) $i = i + 1$;
5. Report o_j and terminate.

5. Communication cost vs. latency

Recall that at the end of the third phase of the algorithm, the top-k scoring objects are among the objects in the partial statistics list maintained by the coordinator. In addition, the coordinator determined τ_3, a lower bound on the scores of the top-k objects. The tentative upper bound determined by the coordinator for all the objects in the partial statistics list is equal to or exceeds τ_3.

In the fourth phase, the coordinator determines the top-k scoring objects among the objects in the partial statistics list. To perform this phase in a single round-trip interaction between the nodes and the coordinator, the coordinator requests from all the nodes all the local data vectors for all the objects in the list. Once the coordinator has collected complete statistics for the objects, it determines their scores and selects the top-k scoring objects. This single-round approach allows low latency, but requires all the local data vectors for each object.

However, many objects can be eliminated by the coordinator using only a fraction of their local data vectors. Rather than collecting all the local vectors for all the objects in a single round, the coordinator can employ several rounds to gradually add local vectors for each object. After each round the coordinator may raise the lower bound it maintains on the scores of the top-k objects, reduce the tentative upper bound it maintains for each object in the partial statistics list, and prune out objects from the list. We call this process the *incremental elimination* of objects.

The incremental elimination process is performed as follows. In the ith round, for every object o_j in the partial statistics list, the coordinator randomly selects 2^{i-1} nodes that have not reported their local vector for the object. Without loss of generality, we denote these nodes by $p_1, p_2, \ldots, p_l$ (where $l = 2^{i-1}$). The coordinator requests that these nodes send it their local statistics vectors for o_j, and adds these new vectors to the partial statistics list. Note that the tentative upper bounds determined by the nodes $p_1, p_2, \ldots, p_l$ for the object o_j are below τ_3 (since they have not been reported in phase 2), while the tentative upper bound determined by the coordinator for o_j is above τ_3. Since adding the new vectors to the partial statistics list cannot increase an object's tentative upper bound (see Section 4.1.1), the coordinator can eliminate objects whose partial tentative upper bound is below τ_3, without collecting all their local data vectors. In addition, at the end of each phase, the coordinator may have completed collecting all the statistics for some objects, and may improve the lower bound on the score of the top-k objects.

After each round, the coordinator examines the list of objects for which it has complete statistics, and sets τ_i' to be the score of the kth scoring object. Note that $\tau_i' \geq \tau_{i-1}'$. Then, the coordinator determines updated tentative upper bounds for the objects in the partial statistics list. Any object whose tentative upper bound is below τ_i' is removed from the list.

While the incremental execution of the fourth phase increases the latency of the algorithm (since it requires more rounds), it avoids collecting all the local vectors for objects that have been eliminated in early rounds, thus dramatically reducing communication cost. The incremental execution is more significant as the number of nodes scales up. Experimental results show that the incremental implementation of the algorithm reduces the overall communication cost by up to a factor of five beyond the improvement reported in Section 4.1. The number of rounds performed in the fourth phase does not exceed $\log(n)$, where n is the number of nodes.

6. Data updates and scalability

The basic algorithm, presented in Section 3, assumes static data with a single coordinator. In this section we will present two versions of the algorithm which allow to relax these assumptions.

6.1. Changes in vectors

In this setup we assume that local vectors can change. Such changes may affect the top-k result. The next algorithm presents an efficient solution to this problem.

(1) Phases I–IV—the same as in the basic algorithm. After phase IV the user receives the current top-k objects.
(2) Phase V—Let $\vec{x}_{j,r}$ be an updated local statistics vector of object j in node r. Assume for the moment that o_j is not among the previous top-k. Given the new updated vector, node r reports o_j and its vector to the coordinator if one of the following conditions holds:
   (a) If $u_{j,r} \geq \tau_2$.
   (b) If $\vec{x}_{j,r}$ does not dominate $\vec{x}_{\text{ref}}$.
   (c) If o_j is member of the global candidate set and its updated vector is not dominated by its previous value ($\vec{x}^{\text{old}}_{j,r}$).
(3) Phase VI—if $\vec{x}_{j,r}$ was reported in phase V, the coordinator needs to recompute o_j's global score. If o_j is not a member of the global candidate set (compiled during phase III), then the coordinator requests all its local statistics vectors from all nodes; otherwise it already holds its vectors. Given o_j's current global score, the coordinator redetermines the top-k result, and upon changes reports to the user.

Theorem 7. *The above-described algorithm correctly identifies the top-k objects, while supporting local vector updates.*

Proof. We show that if none of the conditions in phase V is satisfied, then the top-k objects have not changed. We need to check two cases. Firstly, assume $\vec{x}_{j,r}$ is not a member of the global candidate sets, $u_{j,r} < \tau_2$ and $\vec{x}_{j,r} \succ \vec{x}_{\text{ref}}$. Since o_j is not a member of the global candidate set, its tentative upper bounds in all nodes are below τ_2. In addition, also in node r the tentative upper bound is below τ_2. Since the reference point should not be changed (since $\vec{x}_{j,r} \succ \vec{x}_{\text{ref}}$), then according to Theorem 1, the global score of object o_j is below τ_2, and it is not a member of the top-k objects. In the second case, o_j is a member of the global candidate set, but the updated value $\vec{x}_{j,r}$ is dominated by the its previous value ($\vec{x}^{\text{old}}_{j,r}$). So o_j's new global vector ($\vec{x}_j$) is dominated by its global previous value ($\vec{x}^{\text{old}}_j$), i.e., $\vec{x}^{\text{old}}_j \succ \vec{x}_j$. Since the scoring function $f()$ is monotonic, $f(\vec{x}^{\text{old}}_j) \geq f(\vec{x}_j)$, and the top-$k$ set will not change. Therefore, the coordinator is updated only in specific cases of local vector updates; in the other cases, the nodes handle the update without requiring any communication. □

The algorithm above assumes that the local statistics vectors of the top-k objects have not changed. Such a change may reduce the threshold value (τ_2) and therefore requires re-execution of the top-k query. However, the probability of this scenario is very low $\left(\frac{k}{m}\right)$ as $k \ll m$.

6.2. Multiple coordinators

In the basic algorithm, all the communication takes place between the single coordinator and the nodes. In case of a very large network, this setup may impose considerable communication overhead on the coordinator. In this section we present a different version of the algorithm, which supports multi-coordinator setups for reducing the maximal communication load incurred by any single node in the system.

Given a system consisting of n nodes, we define additional C nodes, denoted $c_1, \ldots, c_C$, to also act as coordinators. We define a lookup function $h()$ which matches objects to coordinators $h : o_j \rightarrow c_t$ ($h()$ is assumed to assign an equal number of objects to each coordinator). We choose this type of matching (as opposed to assigning each coordinator a subset of nodes), since we are solving a problem defined on the object space, and not the node space. Now we present the following multi-coordinator algorithm:

(1) Phase I—Each node locally determines its top-k scoring objects and reports them, with their local statistics vectors, to the appropriate coordinator, using the lookup function $h()$.
(2) Phase II—Each coordinator computes the partial sum statistics vector for each of the reported objects (i.e., the aggregation of reported objects) and defines its current top-k objects. Each coordinator broadcasts its top-k objects to all C coordinators, and then each coordinator locally calculate the kth lower bound score (as described in Section 3.1). This score is defined as the threshold τ_1.
(3) Phase III—The coordinators broadcast τ_1 to the nodes (note that this is just a single scalar), and each node determines the set of objects whose local tentative bound exceeds τ_1, denoted as its local candidate set. Each object in a local candidate set with its local statistics vector is reported to the object's appropriate coordinator, using $h()$.
(4) Phase IV—Each coordinator joins the nodes broadcasts and defines its *coordinator candidate set* (ccs_t). Note that the global candidate set is the union of these lists, i.e., $gcs = \bigcup_{t=1}^{C}\{\text{ccs}_t\}$. Given the information from the various nodes, each coordinator computes the local tentative upper bounds of each object in its ccs_t, and prunes objects whose value does not exceed the threshold. The coordinators' candidate sets are sent to the nodes.
(5) Phase V—Each node returns to each coordinator the local statistics vector for each object in the coordinator candidate list.
(6) Phase VI—Each coordinator computes the global score of each of the objects in its coordinator candidate list, and sends to all coordinators its top-k objects and their score.
(7) Phase VII—Each coordinator can compute the top-k top scored objects, and return this result to the user.

In case a coordinator maintains a list which is smaller than k, then in phases II and VI, the coordinator will publish all the objects in its list.

Lemma 8. *The threshold τ_1 bounds the value of the kth top scored object.*

Proof. Each node reports k objects to the various coordinators. After phase II each coordinator reports its top-k objects (using the partial statistics vectors) or its entire list (in case the list size is smaller than k). At the end of phase II each coordinator maintains a list of objects of length at least k and at most Ck. According to τ_1's definition, it bounds the value of the k top scoring objects in this list, i.e., there are at least k objects whose scores exceed τ_1, as required. □

Theorem 9. *The algorithm correctly finds the top-k scoring objects.*

Proof. Assume that an object o_j is a member of the top-k set, and that it was not reported by the coordinator at the termination of the algorithm. Therefore, either o_j was not a member in the coordinator's top-k scoring objects in phase VI (contradiction to o_j's definition), or it was not a member in the global candidate set. In this case, the tentative upper bound of o_j in each node was below τ_1. But, according to Lemma 8, τ_1 bounds the value of the top-k objects, contradicting o_j's definition. □

The concept of the lookup function $h()$ and multiple coordinators has been extensively used in DHT (Distributed Hash Table) systems over peer-to-peer networks [42,38]. This approach enables the algorithm to scale and reduce the maximal overhead relative to a single coordinator setup. In addition, it can be applied to setups in which nodes can disconnect and connect. In the future we plan to extend the algorithm to peer-to-peer networks.

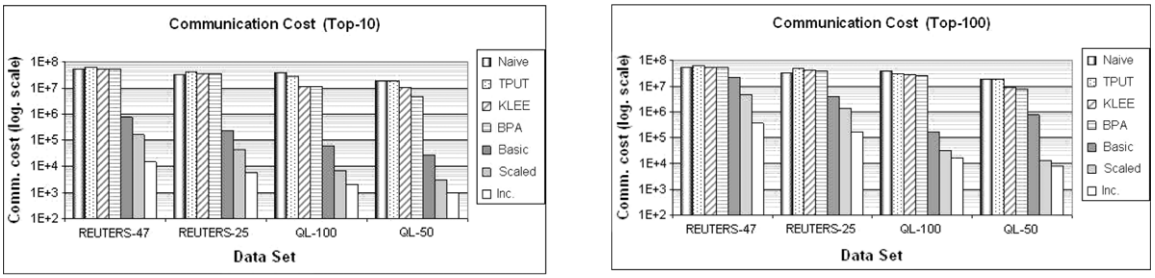


Fig. 6. The communication cost incurred by the variants of our algorithm in comparison with the cost incurred by TPUT, KLEE, BPA, and the basic solution. The communication cost incurred by the incremental variant of our algorithm is two to four orders of magnitude lower than the cost of existing alternatives.

7. Experimental results

We evaluated the proposed algorithm using real-world data compiled from four different data sets. Each set consisted of a collection of tuples, where each tuple contained a list of terms (words). These tuples were partitioned into disjoint groups, simulating data held by distributed nodes. The goal of the experiments was to distributively determine the k most highly correlated pairs of terms in the original collection.

We used *Pearson's Correlation Coefficient* function, defined in detail in Section 1.1. The first two data sets were taken from search queries, collected from a large, centralized search engine, over a period of 3 months [39]. Each tuple in the data represented a search query, and consisted of the search terms that appeared in the query. We simulated data for a set of 100 nodes by splitting the centralized query log in a round robin fashion. The first data set consisted of all 100 nodes and is referred to as the *QL*-100 dataset. The second set, referred to as *QL*-50, consisted of 50 nodes, selected randomly from *QL*-100.

The third and fourth data sets were taken from the Reuters Corpus (RCV1-v2) [37]. This Corpus consists of 804,014 news stories, each tagged as belonging to one or more of 103 content categories. We restricted ourselves to the 47 categories containing at least 10,000 news stories, and used a precomputed list of terms [25] contained in each news story. The third data set, referred to as *RT*-47, consisted of all 47 partitions. The fourth data set, referred to as *RT*-25, consisted of 25 partitions that were randomly selected from *RT*-47. Partitioning by category (as opposed to round robin) yields more heterogeneous data between the nodes.

In our experiments, we compared the communication cost and number of local accesses incurred by our algorithm with those of previously proposed algorithms (TA, TPUT, KLEE). We also evaluated our algorithm's scalability.

7.1. Communication cost

We compared the communication cost incurred by our algorithm with the cost incurred by the Three-Phase Uniform Threshold algorithm (TPUT) [8], KLEE algorithm [29], and Best Position algorithm (BPA) [1]. We tested three variants of our algorithm: the basic variant, which uses spherical bounding regions (as described in Section 4.1); the scaled variant, which uses ellipsoidal bounding regions (as described in Section 4.1); and the incremental variant, which uses ellipsoidal bounding regions and executes the fourth phase incrementally (as described in Section 5). Each algorithm was run on each of the four data sets described above, for $k = 10$ and $k = 100$.

Communication cost was measured in terms of the number of objects (in our case, pairs of terms) transmitted during the execution of the algorithm. For example, each object that is reported to the coordinator in the first phase of our algorithm is counted as a single transmission (and the communication cost of the first phase is therefore kn). However, each object collected by the coordinator in the second phase is counted as two transmissions, one in which the coordinator sends each node a list of requested objects, and the other in which the nodes reply with the local vectors for these objects.

We first compare the basic algorithm with the TPUT, KLEE and BPA algorithms. The results depicted in Fig. 6 show that the communication cost of the basic algorithm is one to two orders of magnitude lower than that of the naive approach. Neither do TPUT, KLEE, and BPA perform well for these top-k queries over our data sets. Their high communication cost is due to the large set of different objects reported to the coordinator (candidate set), emphasizing the problem in local elimination when viewing the scoring function as a dn-dimensional function. In addition, during the execution of these algorithms, the coordinator asks the nodes to complete the statistics vector for every object in the candidate set. Fig. 6 shows that over *RT*-47 and *RT*-25, the cost of this round-trip increases overall communication cost in comparison to the naive approach.

We next tested the scaled and incremental variants of our algorithm. The scaled variant consistently outperformed the basic variant (which uses spherical bounding regions), with communication costs lower by a factor of five. As depicted in Fig. 6, the communication cost of the scaled variant is between one to three orders of magnitude lower than that of the basic solution (one order of magnitude on *RT*-47 and *RT*-25 for the top-100 query, two orders of magnitude on *RT*-47 and *RT*-25 for the top-10 query, and three orders of magnitude on *QL*-100 and *QL*-50 for both top-100 and top-10 queries).

Finally, we tested the incremental variant of our algorithm. The incremental variant consistently outperformed the scaled variant, with communication costs up to four orders of magnitude lower than the basic solution (top-10 query on *QL*-100 and *QL*-50). The results of our experiments indicate that the incremental execution is more effective when the number of nodes is higher. In addition, it was more effective on the *RT* data sets than on the *QL* data sets.

Communication costs are consistently lower for *QL*-100 and *QL*-50 (in comparison with *RT*-47 and *RT*-25). We attribute this to the relatively homogeneous distribution of data among the nodes in these datasets.

7.2. I/O cost

In the third phase of the algorithm every node determines a local candidate set, which consists of all the objects whose tentative upper bound exceeds a threshold value sent by the coordinator. A naive approach for constructing this set is to access every object, determine its tentative upper bound, and then select the ones whose tentative upper bound exceeds the threshold. This approach is inefficient as it requires each node to retrieve all the objects from secondary memory. As explained in Section 4.2, local domination relationships among objects can be used by the nodes to efficiently reduce I/O cost.

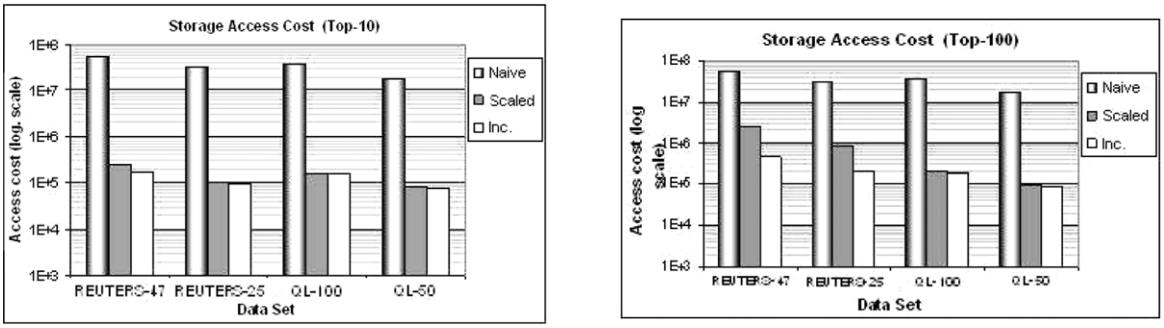


Fig. 7. The I/O cost of constructing the local candidate sets using a progressive detection of skyline objects is an order of magnitude lower than that of the naive solution.

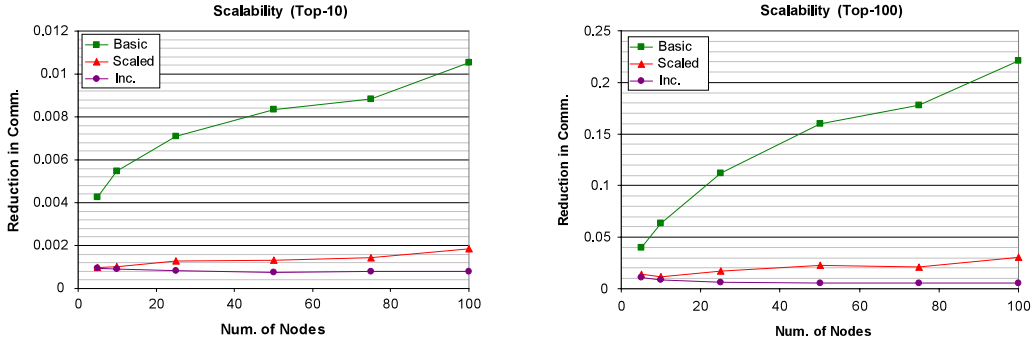


Fig. 8. The graph presents the scalability of the Basic, Scaled, and Incremental algorithms with the number of nodes. The *y*-axis represents the communication cost of the different algorithms divided by the total number of objects over the AOL data. The Incremental algorithm scales well since the communication cost is proportional to the number of nodes. The Basic and Scaled algorithms, however, do not scale well.

Fig. 7 displays the number of objects that have been locally accessed by the nodes using the local domination relationships for the top-10 and top-100 queries. We compared the scaled and incremental algorithms with the naive approach. The results indicate that the method presented in Section 4.2 incurs an I/O cost lower than that of the naive approach by well over an order of magnitude (an average factor of 25 for the top-100 queries, and an average factor of 45 for the top-10 queries). The results also indicate that the incremental algorithm is more effective with increasing value of k. This can be explained by the size of the global candidate list. As the value of k is higher, the global threshold is lower, and more objects are reported to the global candidate list. Thus, the incremental algorithm reduces the number of statistics vectors collected for each object in the global candidate set.

7.3. Scalability

Next we tested the Basic, Scaled, and Incremental algorithms with an increasing number of nodes (from 10 to 100), and computed the communication cost divided by the total number of local statistics vectors. If this ratio is constant, the algorithm scales well. Fig. 8 shows the scalability of these algorithms. The graphs indicate that the Basic and Scaled algorithms do not scale well, while the communication cost of the Incremental algorithm is constantly proportional to the number of nodes. The scalability is achieved by the fourth phase of the Incremental algorithm, which enables the elimination of large sets of objects in the candidate set without collecting their local statistics vectors from all nodes.

8. Conclusion and future work

We presented an algorithm for performing distributed top-k vectorial aggregation queries. The proposed algorithm maintains low latency and low I/O cost. Experiments on real-world data indicate that the algorithm incurs communication costs lower by orders of magnitude than the costs incurred by the existing alternatives, and scales well as the number of nodes participating in the query increases. In the future, we plan to extend the work presented in this paper to handle continuous top-k queries over distributed streams, and to peer-to-peer networks.

References

[1] Reza Akbarinia, Esther Pacitti, Patrick Valduriez, Best position algorithms for top-k queries, in: VLDB 2007, pp. 495–506.
[2] Benjamin Arai, Gautam Das, Dimitrios Gunopulos, Nick Koudas, Anytime measures for top-k algorithms, in: VLDB 2007, pp. 914–925.
[3] Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski, Uwe Thaden, Progressive distributed top-k retrieval in peer-to-peer networks, in: ICDE, IEEE Computer Society, Washington, DC, USA, 2005, pp. 174–185.
[4] Holger Bast, Debapriyo Majumdar, Ralf Schenkel, Martin Theobald, Gerhard Weikum, Io-top-k: index-access optimized top-k query processing, in: VLDB 2006, pp. 475–486.
[5] Stephan Borzsonyi, Konrad Stocker, Donald Kossmann, The skyline operator, pp. 421–430.
[6] Nicolas Bruno, Surajit Chaudhuri, Luis Gravano, Top-k selection queries over relational databases: mapping strategies and performance evaluation, ACM Trans. Database Syst. 27 (2) (2002) 153–187.
[7] Nicolas Bruno, Amelie Marian, Luis Gravano, Evaluating top-k queries over web-accessible databases, in: ICDE 2002, pp. 319–362.
[8] Pei Cao, Zhe Wang, Efficient top-k query calculation in distributed networks, in: PODC 2004, pp. 206–215.
[9] Kevin Chen-Chuan Chang, Seung won Hwang, Minimal probing: Supporting expensive predicates for top-k queries, in: SIGMOD 2002.
[10] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, Gerhard Weikum, Probabilistic ranking of database query results, in: VLDB 2004, pp. 888–899.
[11] Surajit Chaudhuri, Luis Gravano, Evaluating top-k selection queries, in: VLDB 1999, pp. 397–410.
[12] Jan Chomicki, Parke Godfrey, Jarek Gryz, Dongming Liang, Skyline with presorting, pp. 717–719.
[13] Gautam Das, Dimitrios Gunopulos, Nick Koudas, Dimitris Tsirogiannis, Answering top-k queries using views, in: VLDB 2006, pp. 451–462.
[14] Atish Das Sarma, Ashwin Lall, Danupon Nanongkai, Jun Xu, Randomized multi-pass streaming skyline algorithms, Proc. VLDB Endow. 2 (1) (2009) 85–96.
[15] Ronald Fagin, Amnon Lotem, Moni Naor, Optimal aggregation algorithms for middleware, in: PODS 2001, pp. 102–113.
[16] Parke Godfrey, Ryan Shipley, Jarek Gryz, Maximal vector computation in large data sets, in: VLDB 2005, pp. 229–240.

[17] Parke Godfrey, Ryan Shipley, Jarek Gryz, Algorithms and analyses for maximal vector computation, VLDB J. 16 (1) (2007) 5–28.
[18] Ulrich Güntzer, Wolf-Tilo Balke, Werner Kießling, Optimizing multi-feature queries for image databases, in: VLDB 2000, pp. 419–428.
[19] Ulrich Güntzer, Wolf-Tilo Balke, Werner Kießling, Towards efficient multi-feature queries in heterogeneous environments, in: ITCC 2001, pp. 622–628.
[20] Ling Huang, XuanLong Nguyen, Minos N. Garofalakis, Joseph M. Hellerstein, Michael I. Jordan, Anthony D. Joseph, Nina Taft, Communication-efficient online detection of network-wide anomalies, in: INFOCOM, 2007, pp. 134–142.
[21] Ihab F. Ilyas, Walid G. Aref, Ahmed K. Elmagarmid, Joining ranked inputs in practice, in: VLDB 2002, pp. 950–961.
[22] Ihab F. Ilyas, Walid G. Aref, Ahmed K. Elmagarmid, Supporting top-*k* join queries in relational databases, in: VLDB 2003, pp. 754–765.
[23] Ankur Jain, Joseph M. Hellerstein, Sylvia Ratnasamy, David Wetherall, A wakeup call for internet monitoring systems: the case for distributed triggers, 2008.
[24] Donald Kossmann, Frank Ramsak, Steffen Rost, Shooting stars in the sky: an online algorithm for skyline queries, in: VLDB'02, pp. 275–286.
[25] David D. Lewis, Yiming Yang, Tony G. Rose, Fan Li, Rcv1: a new benchmark collection for text categorization research, J. Mach. Learn. Res. 5 (2004) 361–397.
[26] Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, Supporting ad-hoc ranking aggregates, in: SIGMOD 2006, pp. 61–72.
[27] Hua-Gang Li, Hailing Yu, Divyakant Agrawal, Amr El Abbadi, Progressive ranking of range aggregates, Data Knowl. Eng. 63 (1) (2007) 4–25.
[28] Sebastian Michel, Top-*k* aggegation queries in large-scale distributed systems, in: BTW, 2009, pp. 418–427.
[29] Sebastian Michel, Peter Triantafillou, Gerhard Weikum, Klee: a framework for distributed top-*k* query algorithms, in: VLDB 2005, pp. 637–648.
[30] Sebastian Michel, Peter Triantafillou, Gerhard Weikum, Minerva$_\infty$: a scalable efficient peer-to-peer search engine, in: Middleware, 2005, pp. 60–81.
[31] Surya Nepal, M. V. Ramakrishna, Query processing issues in image (multimedia) databases, in: ICDE 1999, pp. 22–29.
[32] Thomas Neumann, Matthias Bender, Sebastian Michel, Ralf Schenkel, Peter Triantafillou, Gerhard Weikum, Distributed top-aggregation queries at large, Distrib. Parallel Databases 26 (1) (2009) 3–27.
[33] Thomas Neumann, Sebastian Michel, Algebraic query optimization for distributed top-*k* queries, in: BTW, 2007, pp. 324–343.
[34] Dimitris Papadias, Yufei Tao, Greg Fu, Bernhard Seeger, Progressive skyline computation in database systems, vol. 30, pp. 41–82.
[35] Dimitris Papadias, Yufei Tao, Greg Fu, Bernhard Seeger, Progressive skyline computation in database systems, ACM Trans. Database Syst. 30 (1) (2005) 41–82.
[36] Byung-Hoon Park, Hillol Kargupta, Distributed Data Mining: Algorithms, Systems, and Applications, Lawrence Erlbaum Associates, 2002.
[37] Tony Rose, Mark Stevenson, Miles Whitehead, The reuters corpus volume 1—from yesterday's news to tomorrow's language resources, in: Proceedings of the Third International Conference on Language Resources and Evaluation, Las Palmas de Gran Canaria, May 2002.
[38] Antony I.T. Rowstron, Peter Druschel, Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems, in: Middleware, 2001, pp. 329–350.
[39] 2002. http://gregsadetsky.com/aol-data/.
[40] Izchak Sharfman, Assaf Schuster, Daniel Keren, A geometric approach to monitoring threshold functions over distributed data streams, ACM Trans. Database Syst. 32 (4) (2007).
[41] Izchack Sharfman, Assaf Schuster, Daniel Keren, Shape sensitive geometric monitoring, in: PODS 2008, pp. 301–310.
[42] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, Hari Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: SIGCOMM, 2001, pp. 149–160.
[43] Kian-Lee Tan, Pin-Kwang Eng, Beng Chin Ooi, Efficient progressive skyline computation, in: VLDB 2001, pp. 301–310.
[44] Martin Theobald, Gerhard Weikum, Ralf Schenkel, Top-*k* query evaluation with probabilistic guarantees, in: VLDB 2004, pp. 648–659.
[45] Panayiotis Tsaparas, Themistoklis Palpanas, Yannis Kotidis, Nick Koudas, Divesh Srivastava, Ranked join indices, in: ICDE 2003, p. 277.
[46] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, Michalis Vazirgiannis, On efficient top-*k* query processing in highly distributed environments, in: SIGMOD 2008, pp. 753–764.
[47] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, Michalis Vazirgiannis, Skyline-based peer-to-peer top-*k* query processing, in: ICDE 2008, pp. 1421–1423.
[48] Tianyi Wu, Dong Xin, Jiawei Han, Arcube: supporting ranking aggregate queries in partially materialized data cubes, in: SIGMOD 2008, pp. 79–92.
[49] Dong Xin, Jiawei Han, Kevin Chen-Chuan Chang, Progressive and selective merge: computing top-*k* with ad-hoc ranking functions, in: SIGMOD 2007, pp. 103–114.
[50] Xifeng Yan, Bin He, Feida Zhu, Jiawei Han, Top-*k* aggregation queries over large networks, in: ICDE, 2010, pp. 377–380.
[51] Ke Yi, Hai Yu, Jun Yang, Gangqiang Xia, Yuguo Chen, Efficient maintenance of materialized top-*k* views, in: ICDE 2003, pp. 189–200.
[52] Hailing Yu, Hua-Gang Li, Ping Wu, Divyakant Agrawal, Amr El Abbadi, Efficient processing of distributed top-*k* queries, in: DEXA 2005, pp. 65–74.
[53] Zhen Zhang, Seung won Hwang, Kevin Chen-Chuan Chang, Min Wang, Christian A. Lang, Yuan-Chi Chang, Boolean + ranking: querying a database by *k*-constrained optimization, in: SIGMOD 2006, pp. 359–370.

Guy Sagy is a graduate student at the Computer Science Faculty, the Technion. His main area of research is distributed algorithms.

Izchak Sharfman recently completed his Ph.D. in the Computer Science Faculty, the Technion. His main area of research is distributed algorithms and geometric methods for stream processing.

Daniel Keren http://cs.haifa.ac.il/~dkeren/, (Ph.D. 1991, Hebrew University in Jerusalem) is currently the chairman of the computer science department in Haifa University, Haifa, Israel. Prof. Keren's main fields of research are geometry and probability. He published mostly in computer vision journals and conferences. Since 2003, he has been working closely with Prof. Assaf Schuster's group in the Technion, in the area of distributed monitoring. His main contribution is in the mathematical aspects of the research such as object modelling, learning, optimization, and probability. A main novelty of the joint research is the incorporation of such mathematical tools into the research paradigm; this allowed to develop entirely new methodologies, based on geometry, to monitor general functions. Prof. Keren's goal is to continue developing the mathematical tools used so far, as well as to develop new ones, to improve and extend the applications of these tools to monitor and mine large, distributed data sets.

Assaf Schuster (http://www.cs.technion.ac.il/~assaf) has established and is managing DSL—the Distributed Systems Laboratory http://dsl.cs.technion.ac.il. Several CS faculty members see DSL as the main scope hosting their applied and systems research, with about 35 graduate and hundreds of undergraduate students working in the lab during the academic year. DSL is supported by Intel, Microsoft, Sun, IBM, and other interested partners. Prof. Schuster is well known in the area of parallel, distributed, high performance, and grid computing. He published over 160 papers in those areas in high-quality conferences and journals. He regularly participates in program committees for conferences on parallel and distributed computing. He consults the hi-tech industry on related issues and holds seven patents. He serves as an associate editor of the Journal of Parallel and Distributed Computing, and IEEE Transactions on Computers. He supervises seven Ph.D. students and ten M.Sc. students, and takes part in large national and EU projects as an expert on grid and distributed computing.