Distributed Threshold Querying of General Functions by a Difference of Monotonic Representation*

Guy Sagy
Computer Science Faculty
Technion
Haifa, Israel
guysagy@cs.technion.ac.il

Daniel Keren
Computer Science Department
Haifa University
Haifa, Israel
dkeren@cs.haifa.ac.il

Izchak Sharfman
Computer Science Faculty
Technion
Haifa, Israel
tsachis@cs.technion.ac.il

Assaf Schuster
Computer Science Faculty
Technion
Haifa, Israel
assaf@cs.technion.ac.il

ABSTRACT

The goal of a *threshold query* is to detect all objects whose score exceeds a given threshold. This type of query is used in many settings, such as data mining, event triggering, and top-k selection. Often, threshold queries are performed over *distributed data*. Given database relations that are distributed over many nodes, an object's score is computed by aggregating the value of each attribute, applying a given scoring function over the aggregation, and thresholding the function's value. However, joining all the distributed relations to a central database might incur prohibitive overheads in bandwidth, CPU, and storage accesses. Efficient algorithms required to reduce these costs exist only for monotonic aggregation threshold queries and certain specific scoring functions.

We present a novel approach for efficiently performing general distributed threshold queries. To the best of our knowledge, this is the first solution to the problem of performing such queries with general scoring functions. We first present a solution for monotonic functions, and then introduce a technique to solve for other functions by representing them as a difference of monotonic functions. Experiments with real-world data demonstrate the method's effectiveness in achieving low communication and access costs.

1. INTRODUCTION

Many tasks require determining the set of objects whose score exceeds a given threshold. Examples include data mining [12, 29], event triggering [30], and top-k selection [11]. We refer to these tasks as *threshold queries*.

Performing threshold queries over distributed database setups can be very challenging. In many distributed setups, data is partitioned over distributed nodes using the same set of attributes across all n nodes[1]. Examples include sensor systems (e.g. NASA EOS [26]), retail databases (e.g. Wal-Mart [26]) and web applications (e.g. Google [27]). In these setups, the score of an object o_j is computed by first aggregating its attribute vectors, denoted $\vec{x}_{j,i}$, to obtain o_j's *global* attribute vector $\vec{x}_j$ ($\vec{x}_{j,i}$ is o_j's attribute vector in the i-th node, so $\vec{x}_j = \frac{1}{n}\sum_{i=1}^{n}\vec{x}_{j,i}$), applying a scoring function $f()$ over the aggregated attributes, and thresholding $f(\vec{x}_j)$ (see Section 3.1). Note that every data attribute is represented in every node. A naive approach to compute such *threshold aggregation queries* is to collect data to a central location. However, that may incur prohibitively high costs, both for local storage accesses at each node and for communication.

Such threshold aggregation functions are more powerful than may appear at first sight; by expanding the object's local data with powers of its attributes, they can be used, for example, to compute two fundamental data characteristics, the variance and maximum functions (for the variance this holds since it can be expressed by the average of the data components and the average of their squares; more details, and a method to approximate the maximum, are in Appendix I). Thresholding the (global) variance over aggregated data is known to be difficult [16].

Aggregation threshold queries have been studied in distributed setups, including distributed association rule mining [9, 29], iceberg queries [39], monitoring [17, 24] and distributed top-k queries [5, 22, 36]. However, these algorithms are applicable only to monotonic scoring function. Alas, in many applications the functions which need to be computed and thresholded are *non-monotonic*, as demonstrated by the following three examples (due to lack of space, full details of these examples are deferred to Appendix II):

EXAMPLE 1. *In a distributed search engine, we wish to find all pairs of search terms whose* global *correlation coefficient is above a certain threshold (meaning that queries which contain "term A" tend to also contain "term B"). Here, "global" can refer to a query log stored in a large distributed data center, or even over distant geographical locations. Generally, finding all pairs of objects whose correlation coefficient is above a certain threshold is a very important task [14, 37]. We shall return to this example in detail in the experiments (Section 7).*

EXAMPLE 2. *A web application stores user's activities over distributed servers, where users are identified by their ips. Each server i contains a database of the number of urls visited by each user j (url_j^i) and the number of advertisements clicked the same user (adv_j^i). A user's global behavior is defined by the average over the local nodes, i.e., $url_j = avg(url_j^i)$ and $adv_j = avg(adv_j^i)$. Given a distance measure $f(url_j, adv_j) = (url_j - A)^2 + (adv_j - B)^2$, where A and B are given values, the application manger can cluster users with similar distance. However, since f is non-monotonic, the local*

*This work was partially supported by the European Commission under ICT-FP7-LIFT-255951 (Local Inference in Massively Distributed Systems).

[1]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.

values can be very different from the global one.

EXAMPLE 3. *We wish to estimate the health of a distributed system, measured by the uniformity of the local traffic and other indicators at the distinct nodes. This can be done by performing PCA (Principal Component Analysis) on a matrix which can be written as an average of local matrices defined at the various nodes [15, 19].*

In all the above examples, the aggregate scoring functions are *non-linear* and *non-monotonic,* and the data is distributed according to our assumptions, i.e. every data attribute is represented in every node.

In this paper we present what constitutes, to the best of our knowledge, the first solution to the problem of performing distributed threshold aggregation queries with *general* (not necessarily linear or monotonic) scoring functions. Our approach is to decompose the query into a set of constraints on the local values held for each object. The global score of objects for which the local constraints are satisfied is guaranteed to be below the threshold. Since the nodes only report objects whose data do not satisfy the local constraints, and since most objects do satisfy them, communication volume is drastically reduced.

We start with an algorithm for monotonic scoring functions, called *TB*(*Tentative Bound*)*-Monotonic*. This algorithm compiles local constraints that minimize communication volume and local accesses. Monotonicity is used to efficiently detect a dominating set and prune dominated objects. An efficient technique for checking whether an object violates its local constraints is also introduced.

Using TB-Monotonic as a building block, we extend the technique to a general method, called the *Tentative Bound* (*TB*) *Algorithm*, for queries that employ general (not necessarily monotonic) scoring functions. We present a novel method for representing scoring functions as a difference of two monotonic functions (*d.m. representation*), which enables to resolve non-monotonic queries. We handle both single variable and multivariate functions; for the latter, we develop a novel method for computing this representation.

Lastly, experiments on real-world data demonstrate the algorithm's effectiveness in achieving low communication and access costs.

1.1 Contributions and Novelty

Distributed geometric constraints were introduced in [30, 31]. This work differs in several important aspects:

- The TB algorithm reduces communication in distributed databases; the algorithms in [30, 31] are suited to distributed streaming setups only.

- TB efficiently reduces the number of objects accessed in each node, as opposed to accessing each object in each node in [30, 31].

- TB is not limited in the number of objects, whereas the algorithms in [30, 31] do not perform well for more than a few thousand objects.

- TB provides an efficient method for verifying whether an object satisfies the local constraints, as opposed to a difficult optimization problem which must be solved in [30, 31].

- While the geometric "bounding theorem" from [30] is applied, it is the only component of previous work used here, and its application is restricted to one phase of the TB-Monotonic algorithm.

- The main technical idea of this paper – an algorithm for querying general functions by representing them as a difference of monotonic functions – is, to the best of our knowledge, entirely new.

The paper is structured as follows. Section 2 discusses related work; section 3 presents notations and terminology, as well as a review of distributed geometric constraints; section 4 present the TB-monotonic algorithm for handling queries defined by monotonic scoring functions; In sections 5 and 6 we present the TB algorithm for general scoring function. Finally in sections 7 and 8 we present experimental results and conclusions.

2. RELATED WORK

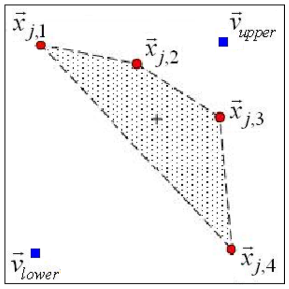
Threshold queries serve as building blocks in many data mining algorithms, including top-*k* queries, association rule mining, and decision tree induction. Cao et al. in [5] have used threshold queries for performing top-*k* algorithm over distributed databases. Their TPUT algorithm consists of three phases: the first is used to determine an estimated value of the k^{th} highest scoring object, the second performs a threshold query which prunes out objects whose score is guarantee to be below the estimated value, and the third searches for the top-*k* within the remaining objects. A lot of work focused on improving the performance of the second phase. Yu et al. [36] improved the threshold query by setting local constraints for each node using the distribution of the node's values. Recently, Michel et al. [22] improved the local constraints using the node's local index. However, these algorithms are limited to monotonic functions only. In comparison, when applying our threshold queries to top-*k* in the general case, we start by collecting the top-*k* scoring objects from each node, and determining the *k*-th among their scores. Then, we query for all objects whose score is above this value, retrieve them, and find the top-*k* objects, which must be in this group; see Section 4.

Other top-*k* algorithms were developed for distributed databases [8, 21] and distributed peer-to-peer setups [3, 23, 32], but they, too, are limited to monotonic functions. Recently, Xin et al. [34] and Zhang et al. [38] suggested new approaches to apply threshold decisions to top-*k* query algorithms defined by non-monotonic scoring functions. However, these approaches are not applicable in a distributed setup.

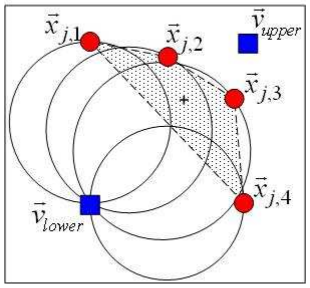
The challenge of performing threshold queries defined by non-monotonic scoring functions was studied for building decision trees over distributed setups. Decision tree induction is an iterative algorithm, where each phase requires determining the highest scoring attribute. This decision can be presented as a top-1 query. For many applications, the score of the best attribute is computed by non-monotonic functions, e.g., Information Gain and Gini Index. For the Gini Index, Giannela et al. [12] have recently showed an efficient technique to determine the best attribute over two nodes, using very little communication. Another approach, presented by Caragea et al. [6], replaced non-monotonic functions with an approximate monotonic functions. Xiong et al. [35] have also replaced the non-monotonic correlation coefficient function with a monotonic function. However, these algorithms are tailored to specific scoring functions.

3. PRELIMINARIES

In this section, we summarize some notations and terminology used in the paper. In addition, we briefly review the definition of local geometric constraints, as presented in [30].



(a) Vectors' convex hull and a visualization of the dominance relationship



(b) Local statistics vector spheres

Figure 1: The convex hull of the local statistics vectors is outlined (a), and it is contained in the union of the local statistics vector spheres, using the lower corner vector as a reference vector (b).

3.1 Notations and Terminology

The system consists of n nodes, p_1, p_2,..., p_n, where each node uses the same set of attributes. A query is initiated on an additional node called the *coordinator*, denoted p_0. There are m objects in the system, denoted o_1,o_2,...,o_m, where various nodes can store data vectors for the same object o_j. The set of attributes for o_j in node i is a d-dimensional vector referred to as *o_j's local statistics vector* and is denoted by $\vec{x}_{j,i}$. The *global statistics vector* for o_j is the average of its local statistics vectors, $\vec{x}_j = \frac{1}{n}\sum_{i=1}^{n}\vec{x}_{j,i}$ (the algorithm can be easily applied for a weighted average as well). For any attribute, we assume that each node maintains its local data structure to allow immediate retrieval of the object with minimal/maximal value for that attribute. Given a scoring function $f()$, the score of object o_j is $f(\vec{x}_j)$.

We say that a vector $\vec{x}$ *dominates* a vector $\vec{y}$, denoted $\vec{x} \succ \vec{y}$, if all the components of $\vec{x}$ are greater or equal than the corresponding components of $\vec{y}$. A function $f()$ is said to be monotonically increasing if $\vec{x} \succ \vec{y} \rightarrow f(\vec{x}) \geq f(\vec{y})$ (this is a natural generalization of the familiar definition for univariate functions).

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 maximal value in each component as the *upper corner vector $\vec{v}_{upper}$*, and the corner with the minimal value in each component as the *lower corner vector $\vec{v}_{lower}$* ($\vec{v}_{upper}$ and $\vec{v}_{lower}$ may not correspond to actual data vectors). Clearly $\vec{v}_{upper} \succ \forall\vec{x}_{j,i} \succ \vec{v}_{lower}$; see Figure 1(a). Assuming that each node allows the immediate retrieval of the objects with minimal/maximal value for each attribute, the coordinator can easily compute $\vec{v}_{lower}$ ($\vec{v}_{upper}$) by collecting these objects from each node in a single communication phase.

3.2 Distributed Geometric Constraints

Given a d-dimensional set of local statistics vectors $\vec{x}_{j,1}$,...,$\vec{x}_{j,n}$ for an object o_j, held by the various nodes, our goal is to define local constraints on $\vec{x}_{j,1}$,...,$\vec{x}_{j,n}$ such that if all of the local constraints are satisfied, the value of the scoring function $f()$ over the global statistics vector, $f(\vec{x}_j)$, is guaranteed not to exceed a given threshold τ.

In [30] continuous monitoring of data streams was addressed, and a geometric method for constructing constraints on the local statistics vectors proposed. Note that $\vec{x}_j$ belongs to the *convex hull* of its local vectors, $\vec{x}_j \in Conv(\vec{x}_{j,1}, \vec{x}_{j,2}, ..., \vec{x}_{j,n})$, as depicted in Figure 1(a). Given an agreed upon common *reference vector*, e.g., $\vec{v}_{lower}$ or $\vec{v}_{upper}$, the constraint on the local statistics vector at each node is defined as follows: the node constructs a sphere centered at the midpoint between $\vec{x}_{j,i}$ and the reference vector ($\vec{v}_{lower}$), with a radius equal to half the distance between them. This sphere is referred to as the *bounding sphere* and is denoted by $B(\vec{v}_{lower}, \vec{x}_{j,i})$. The local constraint of vector $\vec{x}_{j,i}$ is satisfied if the maximum score received by all the vectors that belong to the sphere is below τ: $\max_{\vec{v}\in B(\vec{v}_{lower},\vec{x}_{j,i})} f(\vec{v}) < \tau$. The bounding theorem in [30] guarantees that the union of the bounding spheres constructed by all the nodes will contain the convex hull of the local statistics vectors, i.e., $\text{Conv}(\vec{x}_{j,1}, ..., \vec{x}_{j,n}) \subset \bigcup_{i=1}^{n} B(\vec{v}_{lower}, \vec{x}_{j,i})$ (as depicted in Figure 1(b)). Therefore, if the maximum score received by the vectors that belong to the bounding spheres is below the threshold (i.e., all the constraints are satisfied), then the score received by the global statistics vector is also below the threshold.

4. QUERYING MONOTONIC FUNCTIONS

In this section we present a tentative bound algorithm, called TB-Monotonic, for resolving distributed threshold aggregation queries with monotonic scoring functions. TB-Monotonic uses local constraints, referred to as *tentative upper bounds* (TUBs), to reduce the communication from the nodes to the coordinator and to minimize the local access cost. We first define TUBs and then describe how they are used in TB-Monotonic to reduce communication cost. TB-monotonic will be used as a building block to handle non-monotonic scoring functions.

4.1 Tentative Upper Bound

We now present a method for decomposing distributed threshold queries defined by monotonic scoring functions into a set of local constraints. These constraints are based on the geometric constraints described in section 3.2, but here they are applied in a different way: skyline-related machinery is used to fuse the geometric constraints with the domination relation. The main idea here is that the notion of dominance can be transferred from points to their bounding spheres.

Given a monotonic scoring function $f()$ and a threshold value τ, we want to create a constraint for every object's vector $\vec{x}_{j,i}$. The constraint on the local statistics vector $\vec{x}_{j,i}$ is defined as follows: let $u_{j,i}$ denote the maximum score received by vectors that belong to the bounding sphere defined by $\vec{x}_{j,i}$ and $\vec{v}_{lower}$ (defined in section 3.2), i.e., $u_{j,i} = \max_{\vec{z}\in B(\vec{v}_{lower},\vec{x}_{j,i})} f(\vec{z})$. Note that $u_{j,i}$ can be computed locally, without communication. According to the bounding theorem described in Section 3.2, the global vector of object o_j is contained in the union of the local bounding spheres. Therefore, if $u_{j,i}$ is smaller than τ in every node, then the score for all the vectors in the union of the local bounding spheres is below τ, hence the global score is also below τ. We define the inequality $(u_{j,i} < \tau)$ as the local constraint of object o_j at the node. The value $u_{j,i}$ is referred to as the *tentative upper bound* (TUB) of o_j in node i.

Similarly, we define a set of *tentative lower bounds* (TLB) for an object. The TLB values of o_j are determined by taking the minimum score received by vectors that belong to the bounding sphere defined by $\vec{x}_{j,i}$ and the common reference vector $\vec{v}_{upper}$. This minimum can be quickly approximated to very high accuracy by a simple branch-and-bound algorithm, which makes use of the monotonicity of $f()$. Note that an object's score is guaranteed to be bounded from below by at least one of the TLBs determined for it by the various nodes.

4.2 The TB-Monotonic Algorithm

Following the TUB definition, we now describe the TB-Monotonic algorithm for efficiently performing threshold queries over any monotonic function $f()$.

First, the coordinator computes the common reference vector $\vec{v}_{lower}$, by requesting each node to send its objects with minimal value for each attribute (Section 3.1). Based on this set of received objects, the coordinator determines the global minimal value for each attribute, which define $\vec{v}_{lower}$.

Then, the coordinator sends $\vec{v}_{lower}$ and the threshold query parameters, i.e., the scoring function $f()$ and the threshold value τ, to all nodes. Using $\vec{v}_{lower}$ and $f()$, each node can compile a list of all the objects whose TUB exceeds τ (in the next subsection we show how to do this efficiently). This list is referred to as the *local candidate list* (LCL). One goal is to minimize the size of the LCL. The nodes send the local statistics vectors of the objects in their LCL's to the coordinator; the coordinator then builds the union of the LCL's to form a *global candidate set* (GCS). The TUB's most important feature is to guarantee that the GCS contains all the objects whose global score exceeds τ. Most of the objects, whose score does not exceed τ, are locally pruned by the TUB constraints and do not belong to any LCL.

Finally, the coordinator asks the nodes for the local statistics vectors which it did not yet receive, and which belong to objects in the GCS. This allows to compute the global statistics vectors and global score for all these objects, and to select those whose global score exceeds the threshold.

4.3 Minimizing Local Accesses

The LCL, described in the previous section, contains all the objects whose TUB exceeds a given threshold τ. However, calculating the TUB 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 LCL by computing the TUB 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 whose local statistics vectors are not dominated by any local statistics vector. Given a set of objects in a node p_i, we can therefore bound their TUB value by bounding the TUB value of the skyline objects.

LEMMA 1. *Let $\vec{x}_{a,i}$ and $\vec{x}_{b,i}$ be local statistics vectors for two objects at the node p_i. Furthermore, suppose $\vec{x}_{a,i}$ dominates $\vec{x}_{b,i}$. Then $u_{a,i} \geq u_{b,i}$(the TUB value of the first object is greater than the TUB value of the second object).*

PROOF. The proof of Lemma 1 is given in Appendix III. □

THEOREM 1. *Let S be the set of skyline objects computed over a set O of objects in node p_i and a given threshold τ. If for every object o_j in S, $u_{j,i} < \tau$, then the TUB value for every object in O is below τ.*

PROOF. The proof of Theorem 1 is given in Appendix III. □

Using Theorem 1, we present a progressive algorithm for constructing the LCL, 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 TUB value of each object in this set. If the TUB value of an object $o_j \in S$ is above τ, then o_j is removed from O and added to the LCL. Otherwise, this object is kept in O. The iteration process terminates when the TUB value of all objects in S is below τ. According to Theorem 1, since the TUB value of each object in S is below τ, then the TUB values of all current objects in O are below τ as well. In addition, it is guaranteed that at this point the LCL consists of all the objects whose TUB exceeds τ.

Various algorithms were proposed for efficiently computing the skyline over a database. These algorithm can be classified into two categories: those which do not use pre-processing [4, 13, 10] (and therefore have to make at least one pass over the database) and those which use pre-processing such as indexing and sorting [18, 33, 25]. 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 [25]. 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 (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.

5. QUERYING GENERAL FUNCTIONS

In the previous sections we presented an algorithm for performing distributed threshold aggregation queries for monotonic scoring functions. The monotonicity of the scoring function enabled to exploit domination relationships between objects to reduce communication and local accesses. In this section we propose to resolve queries defined by a general, non-monotonic function, by representing it as a difference of two monotonic (*d.m)* functions. Given a function $f()$, $f() = m_1() - m_2()$ is a *d.m. representation* of $f()$ if $m_1()$ and $m_2()$ are monotonic. A large family of functions – those *of bounded variation* – can be represented in such a manner; these functions include, for example, all functions with bounded derivatives in a certain domain. Also, every continuous function on a closed and bounded domain can be arbitrarily approximated by a function of bounded variation (see Section 6.1 and Appendix IV). This enables the suggested method to deal with most, if not all, functions which need to be computed in real-life applications.

We start by presenting an algorithm that exploits a d.m. representation to efficiently resolve distributed threshold queries. Then, we formally define a large class of functions and show how to construct d.m. representations for its members.

5.1 The TB Algorithm

Given a d.m. representation $f() = m_1() - m_2()$, and a threshold τ, we select an additional threshold referred to as the *dividing threshold,* denoted by t_{div}. The details of how to select t_{div} are discussed in next subsection . We now make the following simple observation, which will serve as the basis for our algorithm: for a vector $\vec{x}$, if $m_1(\vec{x}) < t_{div}$ and $m_2(\vec{x}) > t_{div} - \tau$, then $m_1(\vec{x}) - m_2(\vec{x}) = f(\vec{x}) < \tau$. Therefore, if the global statistics vector $\vec{x}_j$ of the object o_j satisfies these two inequalities, we can conclude that its score is below τ.

Once the dividing threshold has been selected, the coordinator sends τ and t_{div} to all nodes. Upon receiving τ and t_{div}, each node compiles an LCL, which consists of all objects whose tentative upper bound (TUB) on $m_1(\vec{x}_{j,i})$ is above t_{div} (using $\vec{v}_{lower}$ as the reference vector) and all the objects whose tentative lower bound (TLB) on $m_2(\vec{x}_{j,i})$ is below $t_{div} - \tau$ (using $\vec{v}_{upper}$ as the reference vector). The nodes send their LCL's to the coordinator, which compiles a global candidate set, as in the algorithm for monotonic scoring functions. Note that the score of any object that does not appear in the GCS is guaranteed to be below τ. As in the algorithm for monotonic scoring functions, the coordinator requests that the nodes send it the local statistics vectors for all the objects in the

GCS, determines their score, and outputs the objects whose score exceeds τ.

5.2 Defining the Dividing Threshold

The selection of the dividing threshold (t_{div}) may significantly affect the performance of the TB algorithm. Given a scoring function $f()$, a threshold τ, and a d.m. representation of $f()$, t_{div} affects the size of the LCL at each node. Our goal is to select a value for t_{div} that minimizes the size of the global candidate list (i.e., the union of the LCL's determined at the various nodes). In this section we present a simple method for selecting t_{div}. Experiments show that this method produces results that are not too far from those achieved by selecting the optimal value.

We propose a two-phase approach for selecting t_{div}. In the first phase each node determines a tentative value for the local dividing threshold, denoted t_i. This local dividing threshold produces a small LCL. In the second phase an average of the tentative values is determined. This average value is used as the value for t_{div}.

In order to determine t_i at each node, the algorithm scans two sorted lists: L_{TUB}, which holds the local vectors in descending order according to the value of their TUB on $m_1()$, and L_{TLB}, which holds the local vectors in ascending order according to the value of their TLB on $m_2()$. Note that in practice, these lists are computed incrementally on demand, using the skyline approach presented in section 4.3, and their size is bounded according to Theorem 2. The value of t_i is determined by the following iterative algorithm: in the k^{th} iteration, the algorithm determines the vectors in position k from L_{TUB} and L_{TLB}, denoted v'_k and v''_k, respectively. If $v''_k \geq v'_k - \tau$, then we set $t_i = v'_k$ and terminate. Theorem 2 states that the size of the LCL at the i^{th} node produced by using t_i as the dividing threshold is at most twice the size of the smallest possible LCL.

THEOREM 2. *Let X' be the size of the LCL at the i^{th} node produced by using t_i as the dividing threshold. Let t_i^* be the dividing threshold that produces the smallest LCL at the i^{th} node, and let X^* be its size. Then $X' \leq 2X^* + 1$.*

PROOF. Let t_i^* be the optimum local dividing threshold for node i. According to the dividing threshold definition k_1 vectors exist in L_{TUB} whose score exceeds t_i^* ($v'_{k_1} \geq t_i^*$, $v'_{k_1+1} < t_i^*$), and k_2 vectors exist in L_{TLB} whose score is below $t_i^* - \tau$ ($v_{k_2} \leq t_i^* - \tau$, $v''_{k_2+1} > t_i^*$-τ). Note that these vectors are the first values in L_{TUB} and L_{TLB} respectively. Therefore the minimum size of the optimal candidate set is $X^* = \max(k_1, k_2)$. Without loss of generality, we assume that $X^* = k_1$. Next, we show that our algorithm will terminate after at most $k_1 + 1$ iterations. In the $k_1 + 1$ iteration, the value of v''_{k_1+1} is higher than $v'_{k_1+1} - \tau$: Since $v''_{k_1+1} \geq v''_{k_2+1}$($L_{TLB}$ is sorted in ascending order) and $v''_{k_2+1} > t_i^* - \tau$ then $v''_{k_1+1} \geq v''_{k_2+1} > t_i^* - \tau > v'_{k_1+1} - \tau$. Therefore $t_i = v'_{k+1}$ and the algorithm will terminate. If $t_i = v'_{k+1}$ the number of vectors in node i that will be reported is bounded by $2k_1 + 1 = 2X^* + 1$. ☐

6. D.M. REPRESENTATION

In the previous section we assumed that the scoring function can be represented as a difference of two monotonic functions. We now show how to construct such a representation. We start by presenting the notion of the *total variation* of a function.

6.1 Total Variation and the d.m. Representation

The total variation of a univariate function is a well-known measure in real analysis [7]. Intuitively, we can think of the variation of a function $f(x)$ over an interval $[a,b]$ as follows: say the interval represents an interval in time, and the value of the function represents the height of a buoy at a given time. The total variation of the function over the interval, denoted $V_a^b(f)$, is the total distance traveled by the buoy during the time represented by the interval. Formally, the total variation of a multivariate function is defined as follows:

DEFINITION 1. *Let $p = \{\vec{a} = \vec{x}_0 \prec \vec{x}_1 \prec \ldots \prec \vec{x}_n = \vec{b}\}$ be a partition of the multi-dimensional interval $[\vec{a},\vec{b}]$. Let the variation $V(f,p)$ of the multivariate function $f(\vec{x})$ over p be defined as $V(f,p) = \sum_{i=1}^{n} |f(\vec{x}_i) - f(\vec{x}_{i-1})|$, and let $P(\vec{a},\vec{b})$ be the set of all partitions of the interval $[\vec{a},\vec{b}]$. The total variation $V_{\vec{a}}^{\vec{b}}(f)$ of the function $f(\vec{x})$ over the interval $[\vec{a},\vec{b}]$ is defined as the supremum of the variations over all possible partitions: $V_{\vec{a}}^{\vec{b}}(f) = \sup_{p\in P(\vec{a},\vec{b})}(V(f,p))$.*

We say that a function is of *bounded variation* over the interval $[\vec{a},\vec{b}]$ if its total variation is finite. The class of functions of bounded variation over an interval is very large; for example, it contains all functions with bounded derivatives in the interval. Also, most continuous functions are of bounded variation, and those which are not, can be arbitrarily approximated by functions of bounded variation. More on this in Appendix IV.

6.2 Variation Based d.m. Representation

In this section we show how to construct a d.m. representation for any function of bounded variation. We define the d.m. representation $f(\vec{x}) = m_1^*(\vec{x}) - m_2^*(\vec{x})$ on the interval $[\vec{a},\vec{b}]$ by

$$m_1^*(\vec{x}) = \frac{1}{2}(V_{\vec{a}}^{\vec{x}}(f) + f(\vec{x})). \quad (1)$$

$$m_2^*(\vec{x}) = \frac{1}{2}(V_{\vec{a}}^{\vec{x}}(f) - f(\vec{x})). \quad (2)$$

In order to show that this representation is valid, we need to show that $m_1^*(\vec{x})$ and $m_2^*(\vec{x})$ are monotonic.

LEMMA 2. *Let $f(\vec{x})$ be a function over the domain $[\vec{a},\vec{b}]$, and let $\vec{a} \prec \vec{u} \prec \vec{v} \prec \vec{b}$. Then $V_{\vec{a}}^{\vec{v}}(f) \geq V_{\vec{a}}^{\vec{u}}(f) + |f(\vec{v}) - f(\vec{u})|$.*

PROOF. Recall that the total variations $V_{\vec{a}}^{\vec{u}}(f)$ and $V_{\vec{a}}^{\vec{v}}(f)$ are a supremum on the variation of the respective sets $P(\vec{a},\vec{u})$ and $P(\vec{a},\vec{v})$. We map every partition in $P(\vec{a},\vec{u})$ to a partition in $P(\vec{a},\vec{v})$ as follows: let $p_1 = \{\vec{a} = \vec{x}_0 \prec \vec{x}_1 \prec \ldots \prec \vec{x}_n = \vec{u}\} \in P(\vec{a},\vec{u})$. We map p_1 to $p_2 = \{\vec{a} = \vec{x}_0 \prec \vec{x}_1 \prec \ldots \prec \vec{x}_n \prec \vec{x}_{n+1} = \vec{v}\} \in P(\vec{a},\vec{v})$. Clearly, $V(f,p_2) = V(f,p_1) + |f(\vec{v}) - f(\vec{u})|$. Since we mapped *every* member of $P(\vec{a},\vec{u})$ to a member of $P(\vec{a},\vec{v})$ such that the variation of $f(\vec{x})$ on the former exceeds its variation on the latter by $|f(\vec{v}) - f(\vec{u})|$, the total variation of $f(\vec{x})$ on the interval $[\vec{a},\vec{v}]$ exceeds the total variation of $f(\vec{x})$ on $[\vec{a},\vec{u}]$ by at least $|f(\vec{v}) - f(\vec{u})|$, as desired. ☐

Next we prove that $m_1^*(\vec{x})$ and $m_2^*(\vec{x})$ are monotonic:

THEOREM 3. *Let $f(\vec{x})$ be a function over the domain $[\vec{a},\vec{b}]$. Let $m_1^*(\vec{x})$ and $m_2^*(\vec{x})$ be defined as above. Then $m_1^*(\vec{x})$ and $m_2^*(\vec{x})$ are monotonic.*

PROOF. We need to show that $m_1^*(\vec{x})$ is monotonic, i.e., for every $\vec{a} \prec \vec{x}_1 \prec \vec{x}_2 \prec \vec{b}$, $m_1^*(\vec{x}_2) \geq m_1^*(\vec{x}_1)$. According to Lemma 2, $V_{\vec{a}}^{\vec{x}_2}(f) \geq V_{\vec{a}}^{\vec{x}_1}(f) + |f(\vec{x}_1) - f(\vec{x}_2)|$. Using the inequality $|f(\vec{x}_1) - f(\vec{x}_2)| \geq f(\vec{x}_1) - f(\vec{x}_2)$, we obtain $V_{\vec{a}}^{\vec{x}_2}(f) + f(x_2) \geq V_{\vec{a}}^{\vec{x}_1}(f) + f(x_1)$. Employing Eq. (1) we obtain that $m_1^*(\vec{x}_2) \geq m_1^*(\vec{x}_1)$, as desired. Similarly, it follows that $m_2^*(\vec{x})$ is monotonic. ☐

In Figure 2 a representation of $\cos(x)$ as a difference of two monotonic functions is provided.

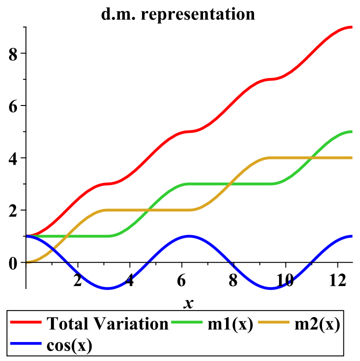


Figure 2: d.m. representation example: $\cos(x) = m_1(x) - m_2(x)$.

6.3 Computing the Total Variation

The total variation of a continuously differentiable univariate function over the interval $[a,b]$ is well-known [7]: $V_{\vec{a}}^{\vec{b}}(f) = \int_a^b |f'(x)| dx$.

The following theorem allows to bound the total variation value for differentiable two-dimensional functions.To the best of our knowledge, it is novel.

THEOREM 4. *Given a differentiable function $f(x,y)$ over the interval $[\vec{a},\vec{b}]$, then $V_{\vec{a}}^{\vec{b}}(f) \leq \int\int_S |\frac{\partial^2 f}{\partial x \partial y}| dxdy + \int_H |\frac{\partial f}{\partial x}| dx + \int_V |\frac{\partial f}{\partial y}| dy$, where S is the rectangular region whose lower left corner is $\vec{a}$, its upper right corner $\vec{b}$, H is the lower horizontal edge, and V the left vertical edge (note that this immediately proves that a function with bounded derivatives if of bounded variation).*

PROOF. The proof of Theorem 4 is given in Appendix IV. □

If computing the integrals is difficult, one may use a dynamic programming algorithm that computes an approximate total variation value (details in Appendix IV).

7. EXPERIMENTAL RESULTS

In this section we evaluate the performance of the TB algorithm for executing distributed threshold queries. We tested the algorithm using three real-world datasets and two scoring functions, which we now review.

7.1 Data Sets and Scoring Functions

We used the Reuters Corpus (RCV1-v2) [28], the AOL Query Log [1] and the Netflix Prize dataset [2]. RCV1-v2 consists of 804,014 news stories, each tagged as belonging to one or more of 103 content categories. We restricted ourselves to the 42 categories containing at least 10,000 news stories, and used a pre-computed list of terms [20] contained in each news story. The AOL data consists of log files collected from a large, centralized search engine over three months. The log files contain the terms present in the search queries. The Netflix data consists of ratings on 17,770 movies by 480,189 anonymous users. A full description of the data sets is in Appendix V.

We used two different correlation functions to test the proposed algorithm, defined over three and four attributes: Pearson's correlation coefficient and chi-square. Querying and thresholding correlation functions is a very frequent task in data mining algorithms [14, 37], such as collaborative filtering, association rule mining, and decision tree construction. The dimensionality (i.e. number of attributes) is similar to that of many aggregation query problems, e.g. [5, 22, 34, 38]. A precise definition of the functions is provided in Appendix V.

7.2 Performance Metrics

We used the following metrics:

Local and Global candidate sets – the number of objects which satisfy the TUB and TLB constraints.

Communication cost – the number of messages sent during the algorithm's execution. For example, each object reported to the coordinator in the first phase of the algorithm (computing $\vec{v}_{lower}$ and $\vec{v}_{upper}$) is counted as a single transmission. Each object collected by the coordinator in the second phase is counted as two transmissions – the coordinator's request and the node's response.

Storage access cost – the number of objects that have been locally accessed by the nodes. We evaluated the reduction in local access cost when using the approach described in section 4.3.

7.3 Algorithms

Naive – The naive solution collects the distributed data and computes the threshold aggregation query in a central location. The communication cost and access cost of this algorithm are equal to the number of local statistics vectors. We use the naive algorithm for comparison since we are not aware of any other algorithm for thresholding general functions in a distributed setting.

TB – The tentative bound (TB) algorithm presented in this paper, using the d.m. representation.

OPC – The Optimal Constraint Algorithm is an offline algorithm used to compare the performance of the TB Algorithm vs. the best that can be achieved using an optimal variants of our approach. The OPC algorithm calculates the convex hull of each object's local statistics vectors and computes the maximum value of the scoring function over this convex hull. It reports to the coordinator only those objects whose maximum value over the convex hull exceeds the given threshold. Thus, OPC determines the theoretical minimal volume of communication that is mandatory for every convex hull-based algorithm, that is, in which the global vector is known to be the average of the local ones.

7.4 Results

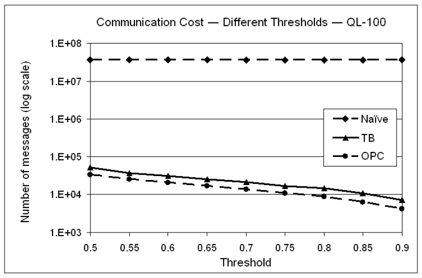
7.4.1 Local and Global Candidate Set

We first tested the efficiency of the TUB and TLB constraints in reducing the local and global candidate sets. We tested these constraints over the three data collections (*QL*-100*, RT*-42*, RC*-50), using Pearson's correlation coefficient as a scoring function with the first two collections, and chi-square with the *RC*-50 data. As Figure 5 demonstrates, these constraints on the average reduce the number of objects in the LCL by three orders of magnitude in comparison to the total number of objects in each node (left two columns). The third column represents the GCS size. As the data distribution is more homogeneous (*QL*-100,*RC*-50), the GCS's size is similar to the LCL's size, relative to more heterogeneous data set (*RT*-42).

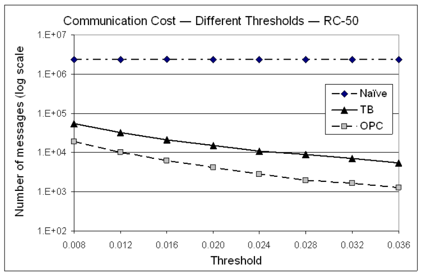
7.4.2 Communion Cost

We compared the communication cost incurred by Algorithm TB with the cost incurred by the naive and OPC algorithms. We tested each algorithm over the three collections of data (*QL*-100*, RT*-42*, RC*-50). We present the performance for a few different threshold values.

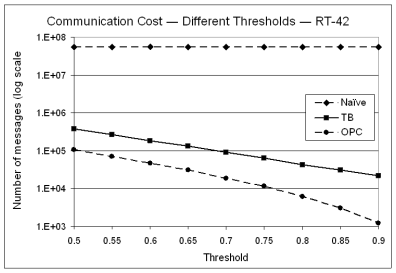
Figure 3 presents the communication cost incurred by the naive, TB and OPC algorithms for different threshold values. In each experiment, the TB communication cost is two to three orders of magnitude lower than the naive approach. The reduction in com-



(a) Communication cost (QL-100)

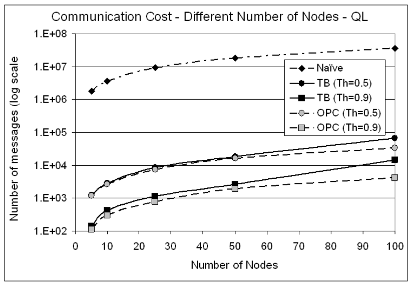


(b) Communication cost (RC-50)

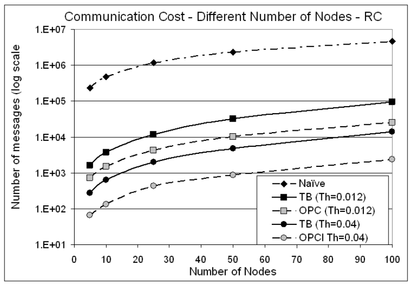


(c) Communication cost (RT-42)

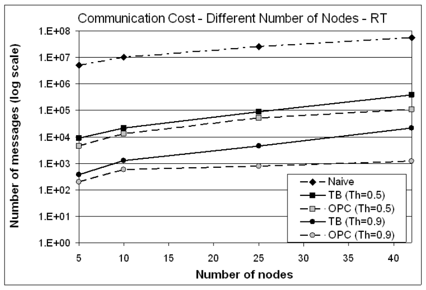
Figure 3: Communication cost (log scale) for different threshold values. TB is two to three orders of magnitude better than the naive algorithm, and at most one order of magnitude worse than OPC.



(a) Communication cost (QL-5..100)



(b) Communication cost (RC-5..100)



(c) Communication cost (RT-5..42)

Figure 4: Communication cost for different numbers of nodes. TB's reduction in communication is maintained when the number of nodes increases.

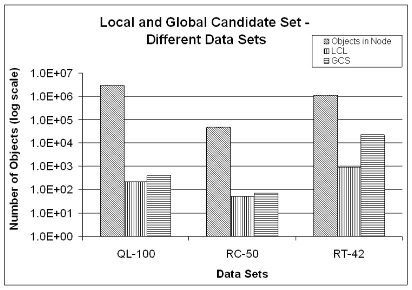


Figure 5: Local and global candidate set size

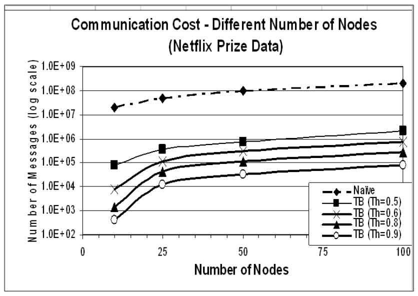


Figure 6: TB algorithm over the Netflix dataset (*NX*)

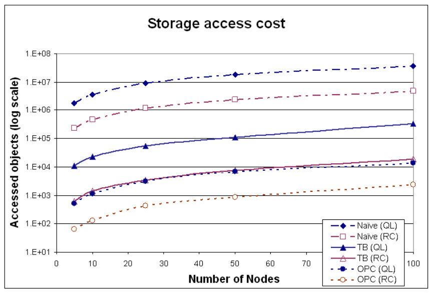


Figure 7: Access costs for the TB algorithm

munication increases with the decrease in threshold values, due to a decrease in the size of the LCL in each node. The results of the TB algorithm are close to the results of the OPC algorithm in *QL*-100 due to the homogeneous distribution of the data.

7.4.3 Scalability

We tested the algorithm's scalability by comparing the communication cost for different numbers of nodes over the three data collections (*QL*-100, *RT*-42, *RC*-50). Figure 4 presents the communication cost incurred by the algorithm for different numbers of nodes. The TB algorithm is scalable as the reduction in communication cost is maintained with the increasing number of nodes (i.e., between two to three orders of magnitude).

We also tested the TB algorithm scalability over the Netflix Prize dataset, using Pearson's correlation coefficient as the scoring function. Figure 6 presents from two to three orders of magnitude reduction in communication relative to the naive algorithm for various numbers of nodes.

7.5 Access cost

Finally, We compared the access cost incurred by the TB algorithm with that incurred by the naive and OPC algorithms. Figure 7 presents access costs for different numbers of nodes over different data collections. TB's access cost is two orders of magnitude lower than the naive approach, and it scales with the number of nodes.

8. CONCLUSIONS

The TB algorithm applies local geometric constraints and a d.m. representation to efficiently solve distributed threshold aggregation queries. Experiments on real-world data demonstrate a reduction in communication cost by two to three orders of magnitude in comparison with the naive method, while minimizing the computational and storage access costs. The main novelty is in the treatment of general scoring functions, by using the d.m. representation. To the best of our knowledge, there is no previous solution for general, non-monotonic functions, for the problems discussed in this paper. In the future, we plan to apply the d.m. representation to other types of database queries which are currently supported only for monotonic functions.

9. REFERENCES

[1] http://gregsadetsky.com/aol-data/.

[2] http://www.netflix.com/.

[3] W. T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive distributed top-k retrieval in peer-to-peer networks. In *ICDE Conf.*, pages 174–185, 2005.

[4] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE Conf.*, pages 421–430, 2001.

[5] P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. In *PODC Conf.*, pages 206–215, 2004.

[6] D. Caragea, A. Silvescu, and V. Honavar. A framework for learning from distributed data using sufficient statistics and its application to learning decision trees. *Int. J. Hybrid Intell. Syst.*, 1(2):80–89, 2004.

[7] N. L. Carothers. *Real Analysis*. Cambridge University Press, 2000.

[8] K. C.-C. Chang and S. won Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *SIGMOD Conf.*, pages 346–357, 2002.

[9] D. W.-L. Cheung and Y. Xiao. Effect of data skewness in parallel mining of association rules. In *PAKDD Conf.*, pages 48–60, 1998.

[10] A. Das Sarma, A. Lall, D. Nanongkai, and J. Xu. Randomized multi-pass streaming skyline algorithms. *Proc. VLDB Endow.*, 2(1):85–96, 2009.

[11] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS Conf.*, pages 102–113, 2001.

[12] C. Giannella, K. Liu, T. Olsen, and H. Kargupta. Communication efficient construction of decision trees over heterogeneously distributed data. In *ICDM Conf.*, pages 67–74, 2004.

[13] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDB Journal*, 16(1):5–28, 2007.

[14] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 2005.

[15] L. Huang, X. Nguyen, M. N. Garofalakis, J. M. Hellerstein, M. I. Jordan, A. D. Joseph, and N. Taft. Communication-efficient online detection of network-wide anomalies. In *INFOCOM conf.*, pages 134–142, 2007.

[16] A. Jain, J. M. Hellerstein, S. Ratnasamy, and D. Wetherall. A wakeup call for internet monitoring systems: The case for distributed triggers. In *HotNets Workshop*, 2004.

[17] R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD*, pages 289–300, 2006.

[18] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. In *VLDB Conf.*, pages 275–286, 2002.

[19] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *SIGCOMM Conf.*, pages 219–230, 2004.

[20] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5(4):361–397, 2004.

[21] A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. In *ICDE Conf.*, pages 369–381, 2002.

[22] S. Michel, P. Triantafillou, and G. Weikum. Klee: a framework for distributed top-k query algorithms. In *VLDB Conf.*, pages 637–648, 2005.

[23] S. Michel, P. Triantafillou, and G. Weikum. Minerva$_\infty$: A scalable efficient peer-to-peer search engine. In *Middleware*, pages 60–81, 2005.

[24] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD Conf.*, pages 563–574, 2003.

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

[26] B.-H. Park and H. Kargupta. Distributed data mining: Algorithms, systems, and applications. Lawrence Erlbaum Associates, 2002.

[27] S. Ramaswamy. Extreme data mining. In *SIGMOD Conf.*, pages 1–2. ACM, 2008.

[28] T. Rose, M. Stevenson, and M. Whitehead. The Reuters Corpus Volume 1 - from Yesterday's News to Tomorrow's Language Resources. In *LREC conf.*, pages 29–31, 2002.

[29] A. Schuster and R. Wolff. Communication-efficient distributed mining of association rules. In *SIGMOD Conf.*, pages 473–484, 2001.

[30] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. *ACM Trans. Database Syst.*, 32(4), 2007.

[31] I. Sharfman, A. Schuster, and D. Keren. Shape sensitive geometric monitoring. In *PODS Conf.*, pages 301–310, 2008.

[32] T. Suel, C. Mathur, J. wen Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. Odissea: A peertopeer architecture for scalable web search and information retrieval. In *WebDB Workshop*, 2002.

[33] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB Conf.*, pages 301–310, 2001.

[34] D. Xin, J. Han, and K. C.-C. Chang. Progressive and selective merge: computing top-k with ad-hoc ranking functions. In *SIGMOD Conf.*, pages 103–114, 2007.

[35] H. Xiong, S. Shekhar, P.-N. Tan, and V. Kumar. Exploiting a support-based upper bound of pearson's correlation coefficient for efficiently identifying strongly correlated pairs. In *KDD*, pages 334–343, 2004.

[36] H. Yu, H.-G. Li, P. Wu, D. Agrawal, and A. E. Abbadi. Efficient processing of distributed top-k queries. In *DEXA Conf.*, pages 65–74, 2005.

[37] J. Zhang and J. Feigenbaum. Finding highly correlated pairs efficiently with powerful pruning. In *CIKM Conf.*, pages 152–161, 2006.

[38] Z. Zhang, S. won Hwang, K. C.-C. Chang, M. Wang, C. A. Lang, and Y.-C. Chang. Boolean + ranking: querying a database by k-constrained optimization. In *SIGMOD Conf.*, pages 359–370, 2006.

[39] Q. Zhao, M. Ogihara, H. Wang, and J. Xu. Finding global icebergs over distributed data sets. In *PODS Conf.*, pages 298–307, 2006.

Appendix I: Computing the Variance and Maximum

Here we show that the variance and maximum can be represented by the aggregation model (a function evaluated at the average data vector). Suppose data is distributed over nodes, and we wish to submit an alert when the data variance crosses a certain threshold. The variance is a fundamental measure of data uniformity, and as such there are numerous applications in which it needs to be monitored [16]. Typically, the exact value of the variance at a given instance is not important; one only needs to submit an alert when it increases beyond a certain threshold. We will now show how to implement this as a threshold aggregation query. Assume for simplicity that the nodes hold scalar values (generalization to data vectors is straightforward). The variance equals the average of the squared values at the nodes, minus the square of the average value. Thus, the nodes can expand their local vectors by adding the squared values, and the variance can be trivially computed from the average of the expanded vectors.

The maximum of the local data vectors is also a very important data characteristic. While it is not possible to exactly describe it as a function on the average vector, it is possible to approximate it with arbitrarily high precision by the following observation: given positive numbers $\alpha_1 \ldots \alpha_m$, $\lim_{n\to\infty}(\alpha_1^n + \ldots + \alpha_m^n)^{\frac{1}{n}} = \max\{\alpha_i\}$. Thus, by choosing a large enough n, and by having the nodes expand their local data by its n-th power, we can arbitrarily approximate the maximum with a function which can be evaluated at the average vector.

Appendix II: Details of the Examples in the Introduction

Example 1:

A *contingency table* of two categorical variables X, Y is a 2×2 table which lists all the possibilities of mutual occurrences of X and Y. Assume for example that X and Y are two terms which may or may not appear in a given document. The contingency table of these two terms is defined as

	X present	X absent
Y present	a	b
Y absent	c	d

(that is, a is the number of documents in which both X and Y are present, etc.). The oft-used *chi-square measure* of the table is defined as $\chi^2 = \frac{(ad-bc)^2}{(a+b)(a+c)(b+d)(c+d)}$. Assume now that we have a database of documents, distributed between many nodes, and we wish to compute the chi-square measure of two terms over the entire database. Each node can compute its own contingency table for the terms, and in the terminology introduced in Section 1, this table is its attribute vector; the global attribute vector is the global contingency table, which is just the sum (or average) of the local ones. Thus, the global score of a pair of terms is the value of the chi-square function evaluated at the sum of the local tables. To demonstrate that this function is non-monotonic, let us look at the following two local contingency tables:

$$\boxed{\begin{array}{c|c} 12 & 4 \\ \hline 3 & 13 \end{array}}, \boxed{\begin{array}{c|c} 8 & 16 \\ \hline 17 & 9 \end{array}}, \text{ and their sum, } \boxed{\begin{array}{c|c} 20 & 20 \\ \hline 20 & 22 \end{array}}.$$

The chi-square values of the two local tables are 10.16 and 5.13, and that of the sum of the tables is 0.046 – much smaller than the values at the local tables. Now, assume two local tables:

$$\boxed{\begin{array}{c|c} 11 & 1 \\ \hline 2 & 1 \end{array}}, \boxed{\begin{array}{c|c} 1 & 3 \\ \hline 1 & 17 \end{array}}, \text{ so that their sum is } \boxed{\begin{array}{c|c} 12 & 4 \\ \hline 3 & 18 \end{array}}.$$

The scores of the local tables are 1.3 and 1.49, and the score of the sum table is 13.89. Therefore, the global score can be either much larger or much lower than the local scores, which violates monotonicity.

Example 2:

Various Internet-based service providers (e.g., search engines and recommender systems) store users' activity in order to improve their performance. In this example, an advertisement system stores users' activity in a distributed database, containing a few geographically distributed nodes. Each node i logs the number of urls visited by each user j (url_j^i) and the number of clicked advertisements (adv_j^i) which, in the terminology of Section 1, are the attribute vectors at the nodes. A user's global activity is defined by the **average** value of the local values, i.e., $url_j = avg(url_j^i)$ and $adv_j = avg(adv_j^i)$. One wishes to retrieve all users whose activity is similar to a given number of clicked urls (A) and clicked advertisements (B), by using the distance function $f(url_j, adv_j) = (url_j - A)^2 + (adv_j - B)^2$. To demonstrate that this function is non-monotonic, assume $A = B = 2$ and that a user's activity is distributed over two nodes. At the first node the local attributes vector is $(url_j^1 = 1, adv_j^1 = 3)$ and at the second it is $(url_j^2 = 3, adv_j^2 = 1)$. The scores of the local vectors are $f(1,3) = f(3,1) = 2$, however the global score is $f(\frac{1+3}{2}, \frac{1+3}{2}) = 0$ – that is, while the local vectors are both at a distance 2 from $A = B = 2$, the average is equal to it. On the other hand, if the local vectors are $(4,4)$ and $(2,2)$, the local scores are 4 and 0, and the global score is 2. Hence, it is generally not possible to bound the global score, given the local ones.

Example 3:

A distributed monitoring paradigm, aimed at detecting volume and other types of anomalies in distributed systems, was introduced in [19]. A low communication implementation was developed in [15]. The basic algorithm proceeds as follows: assume n nodes and m timesteps. The i-th node constructs a vector Y_i of length m, whose j-th entry is the volume of traffic at the j-th timestep at the node. The *health* of the system is measured by aggregating the n vectors $Y_1, \ldots Y_n$ and computing their effective dimension, which is the dimension of the lowest dimensional subspace, denoted F, which well-approximates all the Y_i (typically, one demands that $\sum_{i=1}^{n} ||P_F(Y_i)||^2 \geq (1-\varepsilon) \sum_{i=1}^{n} ||Y_i||^2$, where P_F is the projection operator onto F, and ε a small positive constant, typcially between 0.01 and 0.05). It is well-known that this dimension (denote it d) can be calculated as follows: compute the matrix (denoted A) defined by $\frac{1}{n}\sum_{i=1}^{n} Y_i Y_i^t$, compute A's eigenvalues, ordered by size, $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$, and find the smallest d such that $\frac{\lambda_1^2 + \ldots + \lambda_d^2}{\lambda_1^2 + \ldots + \lambda_m^2} \geq 1 - \varepsilon$. Note that, in the terminology of Section 1, $Y_i Y_i^t$ can be considered as the attribute vector at the i-th node, and the decision on the health of the system is reached by thresholding a function (the effective dimension) of their average. It remains to show that, generally, the effective dimension is non-monotonic. As in Examples 1 and 2, we demonstrate it with two cases. Let us look at the three matrices $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. The effective dimension of each is 1 (eigenvalues of each are 1,0,0) but the effective dimension of their average is 3 (eigenvalues are 1,1,1). To see that the opposite may occur, assume that the i-th node holds e_1 and e_i, where $e_k, k = 1 \ldots n$ are the vectors of the standard basis. Clearly the effective dimension at each node is 2 (two eigenvalues are 1 and the rest are zero). One of the eigenvectors of the global matrix A is equal to 1, and the rest are equal to $\frac{1}{n}$ (the large eigenvalue is due to

e_1, which is present in all nodes, and thus its eigenvalue is n times larger than the ones corresponding to the other e_i's, which are only present in a single node). Thus, the effective dimension tends to 1 as n increases (since the eigenvalues are squared and then added, the total contribution of the small eigenvalues is of order $\frac{1}{n}$). To summarize, the effective dimension of the average matrix can be either larger or smaller than that of the local matrices, proving that it is indeed non-monotonic.

Appendix III: Dominating Bound Theorem

Proof of Lemma 1:

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. It was required that $\vec{v}_{lower}$ be dominated by all the local statistics vectors, thus $\vec{x}_{b,i}$ dominates $\vec{v}_{lower}$. Next, we examine $B(\vec{v}_{lower}, \vec{x}_{a,i})$ and $B(\vec{v}_{lower}, \vec{x}_{b,i})$, the bounding spheres created for o_a and o_b. The centers of these spheres are $\frac{\vec{v}_{lower}+\vec{x}_{a,i}}{2}$ and $\frac{\vec{v}_{lower}+\vec{x}_{b,i}}{2}$ 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, $u_{j,i} = \max_{\vec{v} \in B(\vec{v}_{lower}, \vec{x}_{j,i})}(f(\vec{v}))$.

We claim that for any monotone scoring function $f()$, $u_{a,i} \geq u_{b,i}$. Let $\vec{z} \in B(\vec{v}_{lower}, \vec{x}_{b,i})$ 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 $\vec{z'} \in B(\vec{v}_{lower}, \vec{x}_{a,i})$, such that $\vec{z'}$ dominates $\vec{z}$. Since $\vec{z'}$ belongs to $B(\vec{v}_{lower}, \vec{x}_{a,i})$, 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 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 $\vec{z'} = \vec{c}_{a,i} + \delta(\vec{z})$.

Figure 8 illustrates the constructs described above. It depicts the reference vector, $\vec{v}_{lower}$ (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 $B(\vec{v}_{lower}, \vec{x}_{b,i})$, and the corresponding vector $\vec{z'}$ in $B(\vec{v}_{lower}, \vec{x}_{a,i})$.

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 $B(\vec{v}_{lower}, \vec{x}_{b,i})$, then $\vec{z'}$ belongs to $B(\vec{v}_{lower}, \vec{x}_{a,i})$, as desired. In addition, note that we can express $\vec{z}$ and $\vec{z'}$ as follows: $\vec{z} = \vec{c}_{b,i} + \delta(\vec{z})$ and $\vec{z'} = \vec{c}_{a,i} + \delta(\vec{z})$.

Since $\vec{c}_{a,i}$ dominates $\vec{c}_{b,i}$ then $\vec{z'}$ dominates $\vec{z}$, as desired. Therefore, it follows that $u_{a,i} \geq u_{b,i}$. $\square$

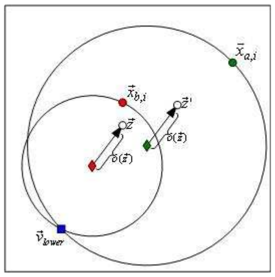


Figure 8: Two local vectors $\vec{x}_{a,i}$ and $\vec{x}_{b,i}$ are depicted, such that $\vec{x}_{a,i}$ dominates $\vec{x}_{b,i}$. The tentative upper bound defined by $\vec{x}_{a,i}$ is guaranteed not to be smaller than that defined by $\vec{x}_{b,i}$.

Proof of Theorem 1:

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 $u_{j,i} < \tau$. If $o_j \notin S$, then according 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 1) and $u_{k,i} < \tau$, then $u_{j,i} < \tau$. $\square$

Appendix IV: Proofs, Algorithms, and Concepts Relating to Total Variation

Proof of Theorem 4:

PROOF. To generalize the computation of the total variation of univariate functions to functions of two variables, we apply *Green's Theorem,* which states that for a domain D with boundary C, and two functions P, Q, the following holds: $\int\int_D \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) = \int_C (Pdx + Qdy)$. For any function f, we then have $\int\int_D \frac{\partial^2 f}{\partial x \partial y} = \int_C \frac{\partial f}{\partial y}dy$. Taking D to be the rectangle in Figure 9a and C its boundary yields $\int\int_D \frac{\partial^2 f}{\partial x \partial y} = f(p_1) + f(p_3) - f(p_2) - f(p_4)$. Let us now look at a typical sum among the ones which define the total variation of f over a rectangle S: $|f(p_5) - f(p_1)| + |f(p_7) - f(p_5)|$ (see Figure 9b). We have $f(p_5) - f(p_1) = [f(p_5) + f(p_1) - f(p_2) - f(p_6)] + [f(p_2) - f(p_1)] + \quad [f(p_6) - f(p_1)] = \int\int_A \frac{\partial^2 f}{\partial x \partial y} + \int_{p_1}^{p_2} \frac{\partial f}{\partial x} + \int_{p_1}^{p_6} \frac{\partial f}{\partial y}$. Similarly, $f(p_7) - f(p_5) = [f(p_7) + f(p_5) - f(p_8) - f(p_4)] + [f(p_4) + f(p_2) - f(p_5) - f(p_3)] + [f(p_8) + f(p_6) - f(p_5) - f(p_9)] + [f(p_3) - f(p_2)] + [f(p_9) - f(p_6)] = \int\int_C \frac{\partial^2 f}{\partial x \partial y} + \int\int_B \frac{\partial^2 f}{\partial x \partial y} + \int\int_D \frac{\partial^2 f}{\partial x \partial y} + \int_{p_2}^{p_3} \frac{\partial f}{\partial x} + \int_{p_6}^{p_9} \frac{\partial f}{\partial y}$. Combining these two identities and using the well-known inequality $|\int g| \leq \int |g|$ (which holds for every function g), as well as the triangle inequality, yields that $|f(p_5) - f(p_1)| + |f(p_7) - f(p_5)| \leq \int\int_S |\frac{\partial^2 f}{\partial x \partial y}| + \int_{p_1}^{p_3} |\frac{\partial f}{\partial x}| + \int_{p_1}^{p_9} |\frac{\partial f}{\partial y}|$. $\square$

This can be extended to prove that this is an upper bound for the total variation of f over S. Note the resemblance of this expression to the bound of a univariate functions: in both cases the bound is expressed as an integral over the absolute values of the derivatives over the domain in which the total variation is computed. This idea can be extended to higher dimensions.

Approximating the total variation

It may be that the integrals which need to be computed in Theorem 4 cannot be exactly evaluated. In this case, one can use a discrete

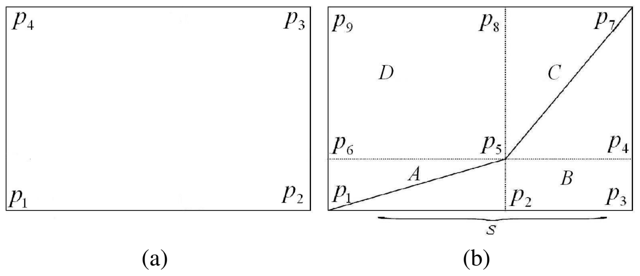


Figure 9: Computing total variation of functions in two variables.

```
TOTALVARIATIONAPPROXIMATION($f$,$p_x$,$p_y$,$G$):
// Init
  $G(x_0,y_0) = f(x_0,y_0)$
  FOR $i = 1$ TO $N_1$ DO
    $G(x_i,y_0) = G(x_{i-1},y_0) + |f(x_i,y_0) - f(x_{i-1},y_0)|$
  FOR $j = 1$ TO $N_2$ DO
    $G(x_0,y_j) = G(x_0,y_{j-1}) + |f(x_0,y_j) - f(x_0,y_{j-1})|$
// Compute $G$ Values
  FOR $i = 1$ TO $N_1$ DO
    FOR $j = 1$ TO $N_2$ DO
      $val_1 = G(x_{i-1},y_j) + |f(x_i,y_j) - f(x_{i-1},y_j)|$
      $val_2 = G(x_i,y_{j-1}) + |f(x_i,y_j) - f(x_i,y_{j-1})|$
      $G(x_i,y_j) = max(val_1,val_2)$
    DONE
  DONE
RETURN $G$;
```

Figure 10: A dynamic programming algorithm to approximate the total variation values.

approximation to the total variation. We now present a dynamic programming algorithm that computes an approximate total variation value. We describe the solution for the two-dimensional case, but it can easily be generalized to any dimension.

Given a function $f(x,y)$ over the range $[(a_x,a_y)..(b_x,b_y)]$, we want to approximate the value of $V_{(a_x,a_y)}^{(x,y)}(f) = \sup_{p \in P((a_x,a_y),(x,y))}(V(f,p))$. Let $p_x = \{a_x = x_0 \prec x_1 \prec ... \prec x_{N_1} = b_x\}$ be a partition of the x axis and $p_y = \{a_y = y_0 \prec y_1 \prec ... \prec y_{N_2} = b_y\}$ a partition of the y axis. Using these partitions, we construct a grid, $G = [p_x \times p_y]$. Using the dynamic programming algorithm described in Figure 10, we approximate the total variation at each of G's vertices (x_i,y_j). The computational complexity is $O(N_1 N_2)$. As N_1 and N_2 increase, G's values are closer to the total variation; however, the computational complexity increases. An example of the total variation of a two dimensional function computed using the dynamic programming algorithm is provided in Figure 11.

The class of functions of bounded variation

The family of functions of bounded variation is broad enough to cover most, if not all, cases of practical interest. For example, every function with bounded derivatives in a closed domain (e.g. a closed box) is of bounded variation; this follows from Theorem 4. The class of such functions covers, for examples, all multivariate polynomials. This is especially important since, according to the famous *Stone-Weierstrass Theorem*, every continuous function in a closed and bounded domain can be arbitrarily approximated by multivariate polynomials. Since the d.m. representation can represent any multivariate polynomial, it can arbitrarily approximate

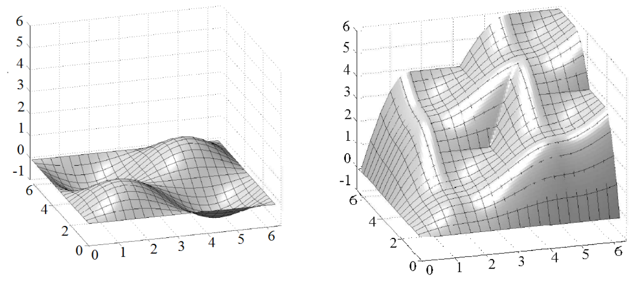


Figure 11: The total variation of a multivariate function. Left: function, right: total variation.

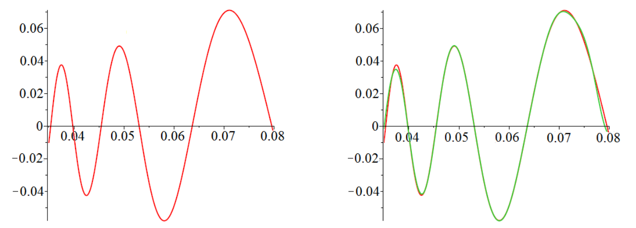


Figure 12: Left: a part of the graph of $x\sin(\frac{1}{x})$. Right: the graph on top (red) super-imposed with a polynomial approximation (green).

any continuous function – even if it is not of bounded variation. Consider for example the function $f(x) = xsin(\frac{1}{x})$ over the interval [0,1] (where $f(0)$ is defined to be 0). This example embodies the class of functions which may not be of bounded variation: such functions must either be unbounded, or have an infinite number of orientation changes, i.e. they have to "zigzag" an infinite number of times; this is because the total variation is the sum of vertical displacements the function's graph makes. Moreover, the sum of the magnitude of these "zigzags" must diverge to infinity. This happens in the case of $xsin(\frac{1}{x})$ since the sum of its vertical changes, or "zigzags", is equal to the harmonic series. We are not aware of any function of practical interest which has this property. It's interesting to note that even this rather pathological example – being continuous – can be arbitrarily approximated by a polynomial, hence, up to a an arbitrarily small error, it can be approximated by a difference of monotonic functions (see Figure 12).

Lastly, let us note that continuity (and, of course, differentiability) are not necessary conditions for bounded variation; for example, every function which is piecewise continuous and with a finite number of jump discontinuities is also of bounded variation.

Appendix V: Datasets and Monitored Functions

We used these datasets to construct four data collections:

QL-100 – we simulated data for a set of 100 nodes by splitting the centralized query log in round robin fashion. Each node i contained 360K vectors representing pairs of search terms A,B and consisting of f_A^i, f_B^i, f_{AB}^i, which denote the number of occurrences of wordA, wordB, and wordA with wordB in the node. The goal was to find the pairs of terms for which Pearson's correlation coefficient exceeds a given threshold. Other numbers of nodes were simulated (*QL-5, QL-10, QL-25* and *QL-50*).

RC-50 – we partitioned the Reuters Corpus to 50 nodes, such that each node contained 16,000 news stories. Each node contained 50K vectors, where the vector corresponding to a word-category pair consisted of their local contingency table. The goal was to find terms whose chi-square measure vis-a-vis the category "CCAT" exceeds a certain threshold. Other numbers of nodes were simulated (*RC-5, RC-10, RC-25,* and *RC-100).*

RT-42 – we divided the Reuters Corpus into 42 partitions according to the different categories (if a story was tagged as belonging to more than one category, we randomly selected one of them). Each partition simulated data held by one node. Each node contained 1M vectors, each consisting of (f_A^i, f_B^i, f_{AB}^i), as in *QL-100*. We queried for pairs of terms whose correlation exceeded a given threshold. While the data in *QL-100* and *RC-50* is homogeneously distributed among the nodes, the *RT-42* data is inhomogeneous (data is homogeneous if an object's local statistics vectors over the different nodes have similar values). Other numbers of nodes were simulated (*RT-5, RT-10* and *RT-25*).

NX-100 – we divided the Netflix data to 100 nodes. Each node consists of a set of randomly chosen users. Each node contained 2M vectors representing pairs of movies, where each vector correspond to the number of viewers which rate each movie individually and the number of viewers which rate both movies. Other numbers of nodes were simulated (*NX-10, NX-25* and *NX-50*).

Functions

The following functions were used in the experiments:

Pearson's Correlation Coefficient – given objects A, B, Pearson's correlation coefficient estimates the relation between them. For example, in the query log datasets, let $f_A^i(f_B^i)$ be the number of queries in node i containing term $A(B)$, and f_{AB}^i the number of queries containing both A and B. The local statistics vector at the i^{th} node is $\vec{x}_{AB,i} = (f_A^i, f_B^i, f_{AB}^i)$. Pearson's correlation coefficient is $\rho(\vec{x}_{AB}) = \frac{f_{AB} - f_A f_B}{\sqrt{(f_A - f_A^2)(f_B - f_B^2)}}$, where $\vec{x}_{AB} = (f_A, f_B, f_{AB})$ is the global statistics vector.

Chi Square – please see Appendix II.