Continuous Fragmented Skylines over Distributed Streams

Odysseas Papapetrou and Minos Garofalakis

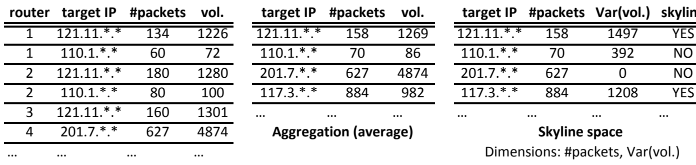
Technical University of Crete

{papapetrou, minos}@softnet.tuc.gr

***Abstract*—Distributed skyline computation is important for a wide range of application domains, from distributed and web-based systems to ISP-network monitoring and distributed databases. The problem is particularly challenging in dynamic distributed settings, where the goal is to efficiently monitor a continuous skyline query over a collection of distributed streams. All existing work relies on the assumption of a single point of reference for object attributes/dimensions, i.e., objects may be vertically or horizontally partitioned, but the accurate value of each dimension for each object is always maintained by a single site. This assumption is unrealistic for several distributed monitoring applications, where object information is *fragmented* over a set of distributed streams (each monitored by a different site) and needs to be aggregated (e.g., averaged) across several sites. Furthermore, it is frequently useful to define skyline dimensions through complex functions over the aggregated objects, which raises further challenges for dealing with object fragmentation. In this paper, we present the first known distributed approach for *continuous fragmented skylines*, namely distributed monitoring of skylines over *complex functions* of *fragmented* multi-dimensional objects. We also propose several optimizations, including a new technique based on random-walk models for adaptively determining the most efficient monitoring strategy for each object. A thorough experimental study with synthetic and real-life data sets verifies the effectiveness of our approach, demonstrating order-of-magnitude improvements in communication costs compared to the only available centralized solution.**

I. Introduction

Since the introduction of the skyline operator [1], the problem of efficiently constructing skylines in distributed environments (such as, client-server and P2P architectures) has been widely studied (see [2] for a survey). The bulk of this work has typically focused on *one-shot* skyline computation, proposing CPU- and communication-efficient strategies for one-time computation of the set of skyline (i.e., dominating, or, Pareto-optimal) objects across *static*, distributed multi-dimensional object collections. Such one-shot techniques over static data are inadequate for new, rapidly-emerging classes of large-scale event monitoring applications, which need to effectively manage, query, and analyze large collections of *distributed data streams*. Prototypical examples include ISP network-monitoring systems (where usage information from a multitude of monitoring points must be tracked and correlated in order to quickly react to hot spots, floods, failures, and attacks), and wireless sensor networks (where multiple remote sensor measurements must be monitored and analyzed for trends, patterns, intrusions, or other adverse events). Querying in such systems is naturally distributed (i.e., over a collection of remote sites), and also *continuous*, that is, we require real-time monitoring of query answers and events, not merely one-shot responses to sporadic queries.



router	target IP	#packets	vol.
1	121.11.*.*	134	1226
1	110.1.*.*	60	72
2	121.11.*.*	180	1280
2	110.1.*.*	80	100
3	121.11.*.*	160	1301
4	201.7.*.*	627	4874
...	...	...	...

target IP	#packets	vol.
121.11.*.*	158	1269
110.1.*.*	70	86
201.7.*.*	627	4874
117.3.*.*	884	982
...	...	...

Aggregation (average)

target IP	#packets	Var(vol.)	skyline
121.11.*.*	158	1497	YES
110.1.*.*	70	392	NO
201.7.*.*	627	0	NO
117.3.*.*	884	1208	YES
...	...	...	...

Skyline space

Dimensions: #packets, Var(vol.)

Fig. 1. Monitoring an ISP network: (a) the raw-distributed data, (b) the aggregated data, (c) the skyline space.

The problem of continuous skyline maintenance in such dynamic distributed settings has also been addressed in recent work [3]. Still, that work, as well as all existing work in distributed skyline processing assumes *horizontal* or *vertical partitioning* of the data, i.e, each site maintains a subset of the complete object vectors, e.g., [4], or a subset of the dimensions of all objects [5], [6]. As such, all previous algorithms rely on the fundamental assumption that there exists a single site in the network maintaining the accurate value for each object's dimension. This configuration enables each site to independently apply local, arbitrarily complex, filtering techniques on the observed updates, drastically reducing the network resources. This assumption, however, is unrealistic for a number of real-world, distributed monitoring applications, where the vector corresponding to each object is determined *by aggregating* (e.g., averaging) partial vector values fragmented over many sites.

To make matters worse, the skyline dimensions may be defined through (possibly) *complex, non-linear functions* over the aggregated object vectors. For example, an ISP might be interested in monitoring the skyline of the aggregate packet volume and the (non-linear) variance of the packet sizes routed to each subnet through each of the edge routers. Such complex *functional* skyline queries are, of course, particularly challenging in the case of fragmented objects: each site only has its partial view of the object vector values, and, for non-linear functions like variance, it is *impossible* to estimate the value of the function on the global object vector from the function values computed locally [7].

Example 1. Consider the problem of monitoring the network of a large ISP. A typical configuration involves installing monitoring code at the edge routers of the ISP to collect workload statistics over sliding windows for a set of IP addresses served by the ISP. Skyline queries on the data *aggregated* over all edge routers are powerful tools for network administrators, for instance, to quickly identify problematic IP addresses or interesting network events. For example, the skyline of the average (over all routers) number of packets and transfer volume, per target IP (data shown in Fig. 1(b)), helps an administrator to quickly focus on the IPs under attack. The skyline dimensions can even be defined through complex, non-linear functions on the aggregated data, such as the variance on the workload per IP, collected by the edge routers (Fig. 1(c)) – a key indicator for sites under a DoS attack. Even though the industry standard in routers enables local statistics maintenance, aggregation of the data in order to maintain the skyline space is a challenging task, due to the sheer volume and volatility of the traffic update streams. The problem is only aggravated by the usage of non-linear functions for the definition of the skyline dimensions (e.g., variance), in which case a router observing a local update cannot even predict the direction of the change at the skyline space, e.g., a sudden drop in the transfer volume at one edge router may actually cause an increase of the variance. This calls for a distributed solution for skyline maintenance, where each edge-router monitor can react only to its local updates that potentially invalidate the existing skyline, notifying the central monitor for further analysis.■

Prior Work. Since the proposal of the skyline operator [1], several aspects of skyline computation have been explored, such as, continuous skylines, e.g., [8], [9], [10], functional (or, *dynamic*) skylines [9], subspace skylines [11], and skylines over distributed and P2P networks [2]. Our contribution lies on the intersection of the areas of distributed, functional and continuous skyline queries, with a novel data fragmentation model.

Algorithms for efficiently constructing skylines in P2P and distributed networks have been widely considered in the recent years (see [2] for a recent survey). These algorithms typically rely on three key ideas to reduce the network communication between participants: (1) *Additivity of the Skyline Operator:* The skyline over all remote sites is always a subset of the union of the local skylines computed at each site, e.g., [4]; (2) *Point Filtering:* Representative points, belonging to one or more sites' local skylines, can help other sites effectively reduce their local skylines [5], [6]; and, (3) *Site Filtering:* Compact local site summaries can be used to target neighboring sites that can potentially contribute skyline points [12]. However, at the core of these approaches is the requirement that the value of each dimension for each object is always maintained by a single site, i.e., vertical or horizontal data partitioning, but not fragmentation. Even though both vertical and horizontal data partitioning models hold significant interest for real-life applications (and, in fact, they can also be handled by our work), they are out of the focus of this work. Instead, our contribution is optimized for the case of fragmented data objects, as this arises frequently in a wide range of network-based applications. Furthermore, we focus on continuous skyline queries, and not on one-shot queries.

Perhaps most similar to ours is the work of Zhang et al. for distributed continuous skyline monitoring [3], which relies on installing filters at remote sites to control the updates that need to be sent to the coordinator. The functionality of filters is similar to the one of threshold-crossing queries used in this work. In fact, in the simple case where data is partitioned but not fragmented, and no functions are used for producing the skyline space, our algorithms (without the adaptivity extension) and the one of [3] produce similar types of local constraints, yet, each one following different optimization strategies. Notice however that [3] supports neither fragmented data nor functional skylines, the combination of which is the main focus of this work. Still, some of the ideas of [3], i.e., near-optimal derivation of filters, as well as the sampling-based extension that trades accuracy for performance, can potentially be adapted for the case of fragmented functional skylines, and will be considered in our future work.

Our Contributions. All previous distributed skyline techniques assume either horizontal or vertical partitioning of the database at the sites, which implies that the accurate value of each dimension for each object is known by one of the sites at any time. In this work, we consider the *fundamentally different problem of continuous fragmented skyline queries*, where: (a) each dimension for each object is fragmented over a number of sites, i.e., the actual values of each object are computed by the aggregation (e.g., averaging) of all object's vectors across all sites, and, (b) the skyline space can be further defined through complex arbitrary functions, parameterized by the aggregate values of the objects. Our contributions are summarized as follows:

• We formally define the continuous fragmented skyline problem, and outline the key underlying challenges.

• We present the first known algorithms for efficient processing of continuous fragmented skyline queries, with dimensions defined through possibly complex arbitrary functions over the aggregate vectors. The two algorithms (termed PIVOT and DIRECT) employ different methodologies for *decomposing* the problem to a select set of distributed threshold-crossing queries that are guaranteed to fire when a change in the skyline occurs. These queries can then be monitored efficiently using ideas from the geometric method [13], [7].

• We propose several optimizations that significantly improve the communication efficiency of our fragmented skyline monitoring algorithms. These include techniques for effectively reducing the queries, which can result to substantial network cost reduction, as well as a technique based on random-walk models for adaptively determining the most efficient monitoring strategy for different objects in the system.

• We present a thorough experimental study of our algorithms over both synthetic and real-life data sets. Our experimental results demonstrate substantial performance benefits compared to the (only alternative) centralized solution, which *often exceed two orders of magnitude*.

II. PROBLEM FORMULATION

System Model. We consider a distributed computing environment, comprising a collection of N *remote processing sites* $\mathcal{P} = \{p_1, p_2, \ldots, p_N\}$ and a designated *coordinator site*. Remote sites receive continuous streams of data updates for a collection of n multi-dimensional objects $\mathcal{O} = \{o_1, o_2, \ldots, o_n\}$ that reside in the system (possibly fragmented across multiple sites), while the coordinator is responsible for maintaining answers to continuous user queries posed over the union of remotely-observed streams (across all sites). The (sub)set of sites monitoring object o_j is denoted by $\mathcal{P}(o_j) \subseteq \mathcal{P}$, while $\mathcal{O}(p_i)$ denotes the (sub)set of objects monitored by site p_i. Following earlier work in the area, e.g., [14], [15], [16], our distributed stream-processing model does not allow direct communication between remote sites; instead, a remote site exchanges messages only with the coordinator, providing it with state information on its (locally-observed) streams. Note that such a hierarchical processing model is, in fact, representative of several application domains, including ISP network monitoring and sensor networks.

At time t, the local state of each object o_j at site p_i is captured by a dynamic d-dimensional *local statistics vector* $\vec{v}(o_j, p_i, t)$. The global state of o_j is defined as the average of o_j's local statistics vectors across all sites in $\mathcal{P}(o_j)$, i.e., the *global statistics vector* $\vec{v}(o_j, t) = \frac{1}{|\mathcal{P}(o_j)|} \sum_{p_i \in \mathcal{P}(o_j)} \vec{v}(o_j, p_i, t)$.[1] (To simplify notation, we omit the explicit dependence on time when referring to the current value of local/global vectors.)

Problem Statement. Our goal is to define effective protocols for continuously monitoring distributed skylines over complex functions of fragmented multi-dimensional objects. More formally, assume that the dimensions of our skyline space are defined through a d'-dimensional *function vector* $\boldsymbol{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$, where each dimension $\boldsymbol{f}[k](\vec{v}(\cdot))$ is a possibly complex, non-linear, arbitrary function over the original d-dimensional global statistics vectors of our objects. We define the notion of *functional dominance* (or, $\boldsymbol{f}$*-dominance*) over fragmented data objects as follows. (Wlog., the definition assumes that lower values are preferred for the skyline.)

Definition 1 ($\boldsymbol{f}$-dominance)**.** *Let $\vec{v}(o_i)$, $\vec{v}(o_j)$ denote the global statistics vectors of objects o_i and o_j. We say that o_i* $\boldsymbol{f}$-dominates *o_j (denoted as $o_i \prec_{\boldsymbol{f}} o_j$) if and only if $\boldsymbol{f}[k](\vec{v}(o_i)) \leq \boldsymbol{f}[k](\vec{v}(o_j))$ for all $k = 1, \ldots, d'$, and $\exists k \in \{1, \ldots, d'\}$ such that $\boldsymbol{f}[k](\vec{v}(o_i)) < \boldsymbol{f}[k](\vec{v}(o_j))$.*

The $\boldsymbol{f}$*-skyline* of the set of objects $\mathcal{O} = \{o_1, \ldots, o_n\}$ fragmented over the remote sites $\mathcal{P}$ is then simply defined as the subset of objects in $\mathcal{O}$ that are not $\boldsymbol{f}$-dominated by any other object in $\mathcal{O}$. That is, $o \in \mathcal{O}$ belongs in the $\boldsymbol{f}$-skyline if and only if $\nexists o' \in \mathcal{O}$ such that $o' \prec_{\boldsymbol{f}} o$.

We address the challenging task of continuously maintaining the $\boldsymbol{f}$-skyline over a large collection of fragmented multi-dimensional objects $\mathcal{O}$ that are dynamically updated across multiple remote sites $\mathcal{P}$. Our protocols aim to *minimize communication* across remote sites and the coordinator — a critical requirement in large-scale monitoring systems, owing to either network-capacity restrictions (e.g., in ISP monitoring, where the volumes of collected utilization and traffic data can be huge [17]), or power and bandwidth restrictions (e.g., in wireless sensor networks, where communication overhead is the key factor in determining sensor battery life [18]). It is important to note that the centralized solution that ships all updates to a coordinator can easily introduce network, computation, and power bottlenecks, overwhelming the underlying network infrastructure. Similarly, simplistic solutions based on batch or periodic updates to the coordinator can either cause large amounts of unnecessary network traffic (with no real change in the skyline) or fail to react to important transitions in a timely manner (when the update period is large). Most importantly, such techniques cannot offer useful guarantees on the quality of the skyline between updates. Instead, our proposed algorithms are *reactive* (based on the observed stream of object updates) and guarantee the continuous correctness of the $\boldsymbol{f}$-skyline at the coordinator.

Example 2. Building on the ISP monitoring scenario of Example 1, the set of remote processing sites $\mathcal{P}$ includes all edge routers in the ISP network, which collect workload statistics for all target IP addresses (or, subnets) contained in $\mathcal{O}$. Assume that we want to monitor the 2-dimensional skyline shown in Fig. 1(c) (average number of packets and variance of transfer volume across all routers, per IP address). Since our $\boldsymbol{f}$*-skylines* are defined on averaged global vectors, we rewrite the variance function using the average transfer volume and the average squared transfer volume per IP at all routers. In particular, each router p_j maintains a three-dimensional vector $\vec{v}(o_i, p_j)$ for each IP address o_i: $\vec{v}[0](o_i, p_j)$ stores the count of all observed packets destined for o_i and routed through p_j, $\vec{v}[1](o_i, p_j)$ stores the sum of the packet sizes, and $\vec{v}[2](o_i, p_j)$ stores $(\vec{v}[1](o_i, p_j))^2$. The global statistics vector for each IP address o_i is the average of the local statistics vectors over all routers, i.e., $\vec{v}(o_i) = \sum_{p_j \in \mathcal{P}(o_i)} \vec{v}(o_i, p_j)/|\mathcal{P}(o_i)|$. The desired skyline space is then defined by function $\boldsymbol{f}$: $\boldsymbol{f}[0] = \vec{v}[0](o_i)$, i.e., the identity function of the average number of packets for each IP address, and $\boldsymbol{f}[1] = Var(\{\vec{v}(o_i, p_j) | p_j \in \mathcal{P}(o_i)\})$ $= \sum_{p_j \in \mathcal{P}(o_i)} \frac{\vec{v}[2](o_i, p_j)}{|\mathcal{P}(o_i)|} - \left(\sum_{p_j \in \mathcal{P}(o_i)} \frac{\vec{v}[1](o_i, p_j)}{|\mathcal{P}(o_i)|} \right)^2 = \vec{v}[2](o_i) - (\vec{v}[1](o_i))^2$. ■

Background: The Geometric Method. Our algorithms decompose functional fragmented skyline monitoring to a small set of distributed threshold crossing queries, which can be monitored locally at each site using the geometric method. We now describe the elements of the geometric method needed for this paper. Further details can be found in [7].

The geometric method addresses the basic problem of monitoring *distributed threshold-crossing queries*; that is, monitor whether $f(\vec{v}(o)) < \tau$ or $f(\vec{v}(o)) > \tau$, for any arbitrary, possibly complex, non-linear function $f()$ of a global statistics vector $\vec{v}(o)$ fragmented over N sites, and a fixed threshold τ. The core idea is that, since it is generally impossible to connect the values of $f()$ on the local statistics vectors to the global value $f(\vec{v}(o))$, one can employ geometric arguments to monitor the *domain* (rather than the range) of $f()$.

To initialize the monitoring process, at time t_0 all nodes $p \in \mathcal{P}(o)$ send their local statistics vectors for the object $\vec{v}(o, p, t_0)$ to a coordinator, where the global statistics vector $\vec{v}(o, t_0)$ is computed. This global statistics vector is also called the global estimate vector $\vec{e}(o)$, and is sent to all network nodes. Whenever a node p_j receives a new local value for o, say, at time t, it updates its local statistics vector and checks whether the new value may cause a threshold crossing. For this check, p_j extracts the statistics delta vector $\Delta\vec{v}(o, p_j) = \vec{v}(o, p_j, t) - \vec{v}(o, p_j, t_0)$. The *drift vector* is then defined as $\vec{u}(o, p_j) = \vec{e}(o) + \Delta\vec{v}(o, p_j)$. These vectors can be used to bound the location of the global statistics vector, which, by definition, is guaranteed to lie, within the convex hull formed by the drift vectors of all nodes and $\vec{e}(o)$ [7]. Therefore, by checking that the convex hull does not overlap the inadmissible region (i.e., the region $\{\vec{v} \in \mathbb{R}^2 : f(\vec{v}) > \tau\}$ in Fig. 2) we can guarantee that the threshold has not been violated.

The problem of course is that the drift vectors are distributed across the nodes. Therefore, the global convex hull is unknown to the individual nodes. To transform the global condition into a local constraint, we place a d-dimensional *bounding ball* around each local delta vector, of radius $||\vec{e}(o) - \vec{u}(o, p_j)||/2$ and centered at $(\vec{e}(o) + \vec{u}(o, p_j))/2$ (see Fig. 2). It can be shown that the union of all these balls completely covers the convex hull of the drift vectors [7]. Therefore, as long as the bounding ball constructed individually at each node is *monochromatic*, i.e., it does not overlap with the inadmissible

[1]More generally, the global statistics vector can be defined through any convex combination of the local statistics vectors.

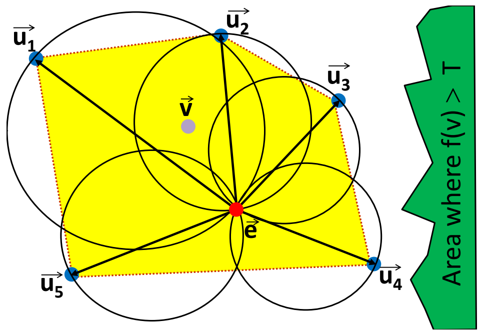


Fig. 2. Estimate vector $\vec{e}$, delta vectors $\Delta\vec{v}(p_i)$ (arrows out of $\vec{e}$), convex hull enclosing the current global vector $\vec{v}$ (dotted outline), and bounding balls $B(\vec{e}, \Delta\vec{v}(p_i))$.

region, the threshold has not been violated, and the node can refrain from sending the local update to the coordinator. If this is not the case, we have a *local threshold violation*, and the site communicates its local $\Delta\vec{v}(p_i)$ to the coordinator. The coordinator then initiates a *synchronization process* that typically tries to resolve the local violation by communicating with some of the sites in order to "balance out" the violating $\Delta\vec{v}(p_i)$. Briefly, this process involves collecting the current delta vectors from (a subset of) the sites, and recomputing the minimum and maximum values of $f(\vec{v})$ according to the new, partial, average. In the worst case, the delta vectors from all N sites are collected, leading to an accurate estimate of the current global statistics vector.

In more recent work, Sharfman et al. [13] show that the local bounding balls defined by the geometric method are special cases of a more general theory of *Safe Zones (SZs)*, which can be broadly defined as *convex subsets of the admissible region* of a threshold query. As long as the local drift vectors stay within such a SZ, the global vector is guaranteed (by convexity) to be within the admissible region of the query [13].

III. Monitoring Fragmented Skylines

In this section, we propose two novel algorithms for continuous fragmented skylines: (1) the *Pivot-Based* (Pivot) algorithm, and (2) the *Direct Monitoring* (Direct) algorithm. Both algorithms rely on effectively *decomposing* the continuous fragmented skyline computation into *a collection of threshold-crossing queries*, which can be efficiently monitored at the participating sites using the geometric method. The main difference between Pivot and Direct lies in the details of this decomposition into threshold-crossing conditions. Still, since both algorithms share a common framework, we describe them in parallel, with references to their particularities.

We start with a brief discussion of the high-level distributed-monitoring protocol. Initially, the user configures the continuous skyline query, by first defining the global statistics vector $\vec{v}$, and, second, the (possibly complex) functions over $\vec{v}$ deriving the skyline dimensions, e.g., variance, L2 norm, or identity function. The system goes through an *initialization phase*, during which the coordinator requests the current local statistics vectors from all sites, and uses them to compute the initial global statistics vectors, the $\boldsymbol{f}$ values for all objects in $\mathcal{O}$, and an initial $\boldsymbol{f}$-skyline, using a standard, centralized algorithm [1]. Then, for each object $o_i \in \mathcal{O}$, the coordinator extracts a set of continuous threshold-crossing queries, denoted as $\mathcal{Q}(o_i)$. While the details of these query sets depend on the employed algorithm (Pivot or Direct), their key property is that they are "safe": *as long as no threshold violation is observed at any site, the skyline is guaranteed not to change*. Finally, the computed global statistics vectors and threshold-crossing queries are shipped to the remote sites observing the corresponding objects, where they are monitored using the geometric method. All updates not violating any threshold query are registered locally at the sites, and only the remaining updates are sent to the coordinator, invoking a synchronization process.

As discussed earlier, a threshold-crossing query focuses on detecting the condition that the value of a function $g()$ over a dynamic vector crosses a fixed threshold value τ. More formally, let t_0 denote the query construction time and let $\vec{v}(t)$ be the dynamic vector; then, using the sign function $\mathrm{sgn}()$, we can define this general threshold-crossing query $Q_{t_0}(g, \vec{v}, \tau)$ as the boolean condition:

$$Q_{t_0}(g, \vec{v}, \tau) \;\equiv\; \mathrm{sgn}(g(\vec{v}(t)) - \tau) \neq \mathrm{sgn}(g(\vec{v}(t_0)) - \tau). \quad (1)$$

Both $g()$ and τ can be multi-dimensional, giving rise to a threshold-crossing query that is equivalent to the OR of the boolean conditions across all dimensions; that is, a threshold crossing along *any* of the dimensions causes the query to fire. To keep our descriptions concise, we employ the multi-dimensional form of Query (1) over our skyline function vector $\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^{d'}$ in the ensuing discussion. Obviously, only the subset of relevant dimensions of $\mathbb{R}^d$ are accounted for monitoring each component function $\boldsymbol{f}[k]$ ($k = 1, \ldots, d'$).

In the remainder of this section, we first explain how the two algorithms extract the threshold-crossing queries for each object. Then, we outline the local monitoring and synchronization processes, which are largely common to both algorithms.

A. Threshold-Crossing Query Decomposition

We now discuss the details of decomposing a continuous fragmented skyline into threshold-crossing queries for both Pivot and Direct. Pivot constructs threshold-crossing queries that pair each object with a set of carefully selected *fixed* pivot points. The purpose of these queries is to ensure that the object remains within a "safe" region, defined by its pivot points in $\mathbb{R}^{d'}$. Direct, on the other hand, constructs threshold-crossing queries that correlate each object with a small set of other *(also moving)* objects from $\mathcal{O}$. The purpose of the queries in this case is to detect when the dominance relation between the objects changes.

We describe the query extraction process, starting with a first approach, where each object monitors its relative positioning with respect to all other objects in the system, resulting in $n - 1$ threshold-crossing queries per object in $\mathcal{O}$. We then propose techniques for *drastically reducing the number of queries* (and, therefore, the network resources) required for effective fragmented skyline monitoring.

The Pivot Algorithm. Pivot constructs threshold-crossing queries that pair an object $o_i \in \mathcal{O}$ with a set of fixed points in the $\mathbb{R}^{d'}$ space, termed *pivot points*. Specifically, during the initialization phase at time t_0, for each pair of objects $\{o_i, o_j\}$, the coordinator computes the pivot point $\overrightarrow{pp}_{i,j}$ as the midpoint between the $\boldsymbol{f}$-values of o_i and o_j, that is, $\overrightarrow{pp}_{i,j} = \frac{1}{2}(\boldsymbol{f}(\vec{v}(o_i, t_0)) + \boldsymbol{f}(\vec{v}(o_j, t_0)))$. Then, it constructs the two threshold-crossing queries: $Q_{t_0}(\boldsymbol{f}, \vec{v}(o_i), \overrightarrow{pp}_{i,j})$ (installed at sites $\mathcal{P}(o_i)$) and $Q_{t_0}(\boldsymbol{f}, \vec{v}(o_j), \overrightarrow{pp}_{i,j})$ (installed at sites $\mathcal{P}(o_j)$). As an example, Fig. 3(a) depicts a sample data set with five 2-dimensional objects, and Fig. 3(b) shows the same

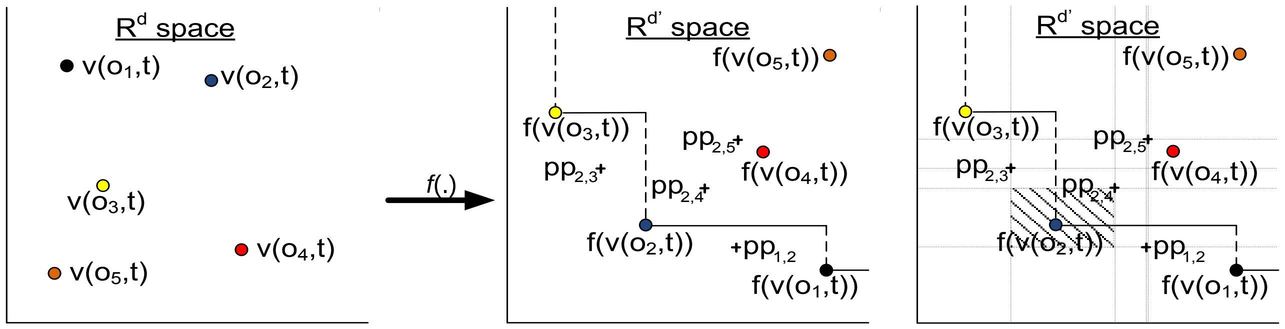


Fig. 3. Pivot-based method: (a) the original $\mathbb{R}^d$ space, (b) the four pivot points for o_2 in the transformed $\mathbb{R}^{d'}$ space, (c) the safe region for o_2.

objects in the $\boldsymbol{f}$-skyline space, including the four pivot points defined for o_2 with respect to all other objects. A site observing o_2 then has to monitor the following threshold-crossing queries (one per pivot point): $Q_{t_0}(\boldsymbol{f}, \vec{v}(o_2), \overrightarrow{pp}_{1,2})$, $Q_{t_0}(\boldsymbol{f}, \vec{v}(o_2), \overrightarrow{pp}_{3,2})$, $Q_{t_0}(\boldsymbol{f}, \vec{v}(o_2), \overrightarrow{pp}_{4,2})$, and $Q_{t_0}(\boldsymbol{f}, \vec{v}(o_2), \overrightarrow{pp}_{5,2})$.

Consider the geometric interpretation of the PIVOT technique. Each pivot point $\overrightarrow{pp}_{i,j}$ partitions the $\mathbb{R}^{d'}$ space into $3^{d'}$ subspaces: three subspaces for each dimension $k = \{1, \ldots, d'\}$, namely, $\{\vec{x} : \vec{x}[k] < \overrightarrow{pp}_{i,j}[k]\}$, $\{\vec{x} : \vec{x}[k] > \overrightarrow{pp}_{i,j}[k]\}$, and $\{\vec{x} : \vec{x}[k] = \overrightarrow{pp}_{i,j}[k]\}$. The intersection of these $3^{d'}$ subspaces across all threshold-crossing queries for object o_i that contains $\boldsymbol{f}(\vec{v}(o_i))$ effectively defines a *safe region* for o_i; that is, as long as $\boldsymbol{f}(\vec{v}(o_i))$ remains in this region, its relative positioning in the skyline with respect to all other objects in $\mathcal{O}$ remains unchanged. For example, Fig. 3(c) depicts the (shaded) safe region for o_2. Note that the threshold-crossing queries installed at $\mathcal{P}(o_i)$ monitor exactly this safe-region condition for o_i. It is not difficult to prove that this scheme is correct: As long as no PIVOT threshold-crossing query fires, the relative positioning of any object pair in the fragmented skyline (i.e., their relative dominance) remains unchanged, and, thus, the previously-computed skyline remains valid.

The DIRECT Algorithm. Rather than placing fixed pivot points somewhat arbitrarily at the midpoint of two objects, DIRECT *directly monitors* the relative dominance relation across each pair of fragmented objects, based on the vector difference of their $\boldsymbol{f}$-values. Formally, consider any pair of objects $o_i, o_j \in \mathcal{O}$ and, for the time being, assume that both objects are observed at exactly the *same subset of remote sites*, i.e., $\mathcal{P}(o_i) = \mathcal{P}(o_j)$. We define the function-difference vector $\boldsymbol{g}(\vec{v}(o_i)|\vec{v}(o_j)) = \boldsymbol{f}(\vec{v}(o_i)) - \boldsymbol{f}(\vec{v}(o_j))$, where $\vec{v}(o_i)|\vec{v}(o_j)$ denotes the concatenation of the objects' global statistics vectors; thus, $\boldsymbol{g} : \mathbb{R}^{2d} \rightarrow \mathbb{R}^{d'}$. Then, for each such object pair, the coordinator simply constructs the threshold-crossing query $Q_{t_0}(\boldsymbol{g}, \vec{v}(o_i)|\vec{v}(o_j), \vec{0})$ and installs it at all sites in $\mathcal{P}(o_i) = \mathcal{P}(o_j)$ to monitor updates to either o_i or o_j ($\vec{0}$ denotes the all-zero d'-dimensional vector). For instance, in our running example in Fig. 3, the set of DIRECT threshold queries extracted for o_2 is $\mathcal{Q}(o_2) = \{Q_{t_0}(\boldsymbol{g}, \vec{v}(o_2)|\vec{v}(o_j), \vec{0}) : j = 1, 3, 4, 5\}$. Once again, it can be formally shown that, as long as none of the DIRECT threshold-crossing queries fires, the fragmented skyline cannot change.

A number of issues with the DIRECT algorithm are worth noting. First, observe that it effectively *doubles the dimensionality* of the local geometric bounding constraints since it needs to account for updates to both objects. This increased dimensionality typically leads to more frequent local threshold violations and higher communication costs. (This issue can be avoided for certain function types, e.g., when $\boldsymbol{f}$ is linear, but not on the general case.) A second, and perhaps more subtle, issue concerns the extension of DIRECT to handle the general case of object pairs $\{o_i, o_j\}$ that are observed at different subsets of the remote sites (i.e., $\mathcal{P}(o_i) \neq \mathcal{P}(o_j)$), and its effectiveness in such settings. To ensure correctness in this case, the DIRECT threshold query over $\vec{v}(o_i)|\vec{v}(o_j)$ needs to be monitored across all sites in $\mathcal{S} = \mathcal{P}(o_i) \cup \mathcal{P}(o_j)$ (with parts of the local statistics vector zeroed out at sites observing only one of the objects). Furthermore, since the geometric method requires the monitored function(s) $\boldsymbol{g}$ to be defined over the average of the local vectors across all $|\mathcal{S}|$ participating sites, an additional weighting step is needed for the local statistics vectors used in the computation of the $\boldsymbol{f}$-values. The key observation here is that the average global statistics vector $\vec{v}(o_i)$ over all sites in $\mathcal{P}(o_i)$ is equal to the average vector over the super-set $\mathcal{S}$ (assuming zero vectors for sites in $\mathcal{S} - \mathcal{P}(o_i)$) multiplied by $|\mathcal{S}|/|\mathcal{P}(o_i)|$. Therefore, we can apply the geometric method assuming that o_i is monitored by all sites in $\mathcal{S}$, by simply scaling each of its local statistics vectors by $|\mathcal{S}|/|\mathcal{P}(o_i)|$ (and, similarly for o_j). This scaling, however, has the adverse effect of increasing the radius of the local bounding ball for the object, thereby increasing the number of local violations. In fact, it can be formally proved that the performance of DIRECT is *worse* than that of PIVOT under certain such settings.

Theorem 1. *Monitoring the* DIRECT *threshold-crossing query* $Q_{t_0}(\boldsymbol{g}, \vec{v}(o_i)|\vec{v}(o_j), \vec{0})$ *for object* o_i *at sites* $\mathcal{S} = \mathcal{P}(o_i) \cup \mathcal{P}(o_j)$ *is provably less communication-efficient than monitoring the corresponding* PIVOT *threshold query* $Q_{t_0}(\boldsymbol{f}, \vec{v}(o_i), \overrightarrow{pp}_{i,j})$*, when all functions in* $\boldsymbol{f}$ *are linear, and* $\frac{|\mathcal{S}|}{|\mathcal{P}(o_i)|} > 2$.

The proof is deferred to the extended version of the paper. Similar results can also be shown for other types of functions. Note that the cardinality ratio condition $\frac{|\mathcal{S}|}{|\mathcal{P}(o_i)|} > 2$ is easily satisfied when objects are monitored by distinct subsets of sites; furthermore, some of the optimizations discussed later in this section (e.g., grouping) further exacerbate this problem for the DIRECT algorithm.

B. Reducing the Number of Queries

The total number of threshold crossing queries influences the network cost of PIVOT and DIRECT, since: (a) all queries need to be sent to the sites, during initialization and after threshold crossings, and, (b) a higher number of queries can obviously lead to tighter safe regions and more frequent threshold crossings. To reduce network cost, we need to extract a sufficient subset of queries that can still guarantee the correctness of the skyline. In this section, we show how the total number of queries can be substantially reduced (from quadratic to linear on the number of objects). It is important to note that this reduction comes without increasing the tightness of the threshold queries, which would have the adverse effect of increasing the frequency of threshold violations and the induced network cost. In fact, the safe regions are, for most objects, substantially relaxed. To avoid repetition, the ensuing discussion focuses primarily on PIVOT. The same optimizations can be adapted for DIRECT in a reasonably straightforward manner.

Eliminating Redundant Threshold Queries. A crucial observation is that not all changes in pairwise dominance relations between objects in $\mathcal{O}$ are important for skyline monitoring. For example, the skyline will not change if o_4 (Fig. 3(b)) is updated such that it no longer $\boldsymbol{f}$-dominates o_5. In fact, there are only two types of threshold-crossing queries that can signify a change in the skyline: (1) Queries monitoring the *domination of a non-skyline object by a skyline object*, where a violation may indicate the entry of a new object in the skyline; and, (2) Queries monitoring the *dominance (i.e., Pareto optimality) of a skyline object*, where a violation may indicate the removal of an object from the skyline. All other queries are essentially redundant and can be safely dropped.

(1) Queries Monitoring Domination of a Non-Skyline Object: The key observation here is that a non-skyline object cannot enter the skyline as long as it is $\boldsymbol{f}$-dominated by at least one skyline object. Thus, for any given non-skyline object o_i, it suffices to monitor a single threshold-crossing query between o_i and a skyline object o_j that $\boldsymbol{f}$-dominates o_i. Having no knowledge on the distribution of future updates, the best threshold condition to monitor is the one that maximizes the minimum distance (slack) between o_i and the resulting pivot point $\overrightarrow{pp}_{i,j}$ along all d' dimensions; that is, we select the skyline object o_j that $\boldsymbol{f}$-dominates o_i and maximizes $\min_{\ell=1}^{d'}\{\boldsymbol{f}[\ell](\vec{v}(o_i))-\boldsymbol{f}[\ell](\overrightarrow{pp}_{i,j})\}$. In our Fig. 3(b) example, this gives rise to threshold queries for the pairs $\{o_2, o_4\}$ and $\{o_2, o_5\}$.

(2) Queries Monitoring Dominance of a Skyline Object: A skyline object o_i may exit the skyline only when some other skyline object o_j moves to $\boldsymbol{f}$-dominate o_i. (A non-skyline object can cause the removal of a skyline object only after itself enters the skyline, thereby causing another threshold query of the previous class to fire.) Furthermore, not all pairs of skyline objects need to be monitored, since some skyline objects impose tighter threshold constraints than others, and will always be violated first. For example, o_1 cannot move to dominate o_3 without first crossing its threshold query with o_2. Specifically, for any skyline object o_i, the coordinator constructs a threshold-crossing query between o_i and all other skyline objects whose $\boldsymbol{f}$ values *immediately* precede or follow $\boldsymbol{f}(o_i)$ along any dimension of the $\mathbb{R}^{d'}$ space. In our Fig. 3(b) example, this gives rise to threshold queries for the pairs $\{o_2, o_3\}$ and $\{o_2, o_1\}$.

Using the above ideas, the total number of threshold-crossing queries in the system is effectively reduced from $\Theta(n^2)$ to (at most) $2(n + s(d'-1))$, where s denotes the size of the skyline (and, typically, $s << n$).

Grouping of Pivot Points. Even after eliminating redundant queries, skyline objects with dense dominance regions may end up participating in a large number of threshold-crossing queries with different pivot points. This translates to high transfer volume for sending all these threshold queries to sites, both during initialization and after threshold crossings. To further reduce PIVOT's resource requirements, the coordinator forms *groups of pivot points* for each skyline object o_i, and replaces each group with a single "composite" pivot point that imposes equivalent threshold constraints on o_i. Threshold queries are then constructed based on the computed composite pivot points, which are much fewer than the original ones, and typically also enable enlarging the safe-zones for the non-skyline objects.

Precisely, the pivot points of each skyline object o_i are grouped based on their relative positioning with respect to $\boldsymbol{f}(\vec{v}(o_i))$ in all d' dimensions. Any two pivot points $\overrightarrow{pp}_{i,j}$ and $\overrightarrow{pp}_{i,k}$ are grouped together if their $\boldsymbol{f}$ values are on the same side of $\boldsymbol{f}(\vec{v}(o_i))$ in all d' dimensions, or, more formally, if $\mathrm{sgn}(\boldsymbol{f}[\ell](\vec{v}(o_i))-\overrightarrow{pp}_{i,j}[\ell]) = \mathrm{sgn}(\boldsymbol{f}[\ell](\vec{v}(o_i))-\overrightarrow{pp}_{i,k}[\ell])$ for all $\ell = 1, \ldots, d'$. All pivot points belonging in the same group $G = \{\overrightarrow{pp}_{i,j}, \overrightarrow{pp}_{i,k}, \ldots\}$ are then replaced by a composite pivot point $\overrightarrow{pp}_{i,G}$, defined as follows:

$$\overrightarrow{pp}_{i,G}[\ell] = \begin{cases} \min_{\overrightarrow{pp}\in G}\overrightarrow{pp}[\ell] & \text{if } \min_{\overrightarrow{pp}\in G}(\overrightarrow{pp}[\ell]) \geq \boldsymbol{f}[\ell](\vec{v}(o_i,t)) \\ \max_{\overrightarrow{pp}\in G}\overrightarrow{pp}[\ell] & \text{if } \max_{\overrightarrow{pp}\in G}(\overrightarrow{pp}[\ell]) < \boldsymbol{f}[\ell](\vec{v}(o_i,t)) \end{cases}$$

for $\ell = 1, \ldots, d'$.

By construction, this composite pivot point imposes the same restrictions on $\boldsymbol{f}(\vec{v}(o_i,t))$ as the collection of pivot points in G. Furthermore, all pivot points in G for objects $\{o_j, o_k, \ldots\}$ are also replaced by the composite pivot point $\overrightarrow{pp}_{i,G}$, which can result in additional slack for these objects, yet without introducing errors. In the example of Fig. 3(c), grouping replaces $G = \{\overrightarrow{pp}_{2,4}, \overrightarrow{pp}_{2,5}\}$ with a single composite pivot point $\overrightarrow{pp}_{2,G}$ that coincides with $\overrightarrow{pp}_{2,4}$, which actually gives additional slack to o_5.

Combining the elimination of redundant queries with pivot-point grouping results in a maximum of $2d'+1$ threshold-crossing queries for each skyline object ($2d'$ pivot points for its neighboring skyline objects and one composite pivot point for all objects in its dominance region). Each non-skyline object only needs to monitor a single threshold query. Thus, the total number of threshold queries in the system is effectively reduced to (at most) $n + 2sd'$, that is, $O(n)$.

The following theorem summarizes the correctness guarantees offered by the resulting queries.

Theorem 2. *The extracted threshold queries are sufficient for accurate fragmented skyline monitoring, i.e., as long as no threshold violation occurs, the fragmented skyline is guaranteed to stay the same. They are also* minimal*, in the sense that omitting any of the queries breaks the correctness guarantees.*

The proof is deferred to the extended version of the paper.

C. Local Monitoring

Note that the threshold-crossing queries produced by our decomposition do not directly translate to local monitoring conditions, since these are defined over the aggregate object values (the global statistics vectors). However, nodes can exploit the geometric method to efficiently monitor these threshold queries without imposing centralization of all updates. Briefly, a node receiving an update for an object o forms the bounding ball (see Section II), and tests for monochromicity w.r.t. all threshold queries. This test is performed by finding the minimum and maximum value of the monitored function inside the bounding ball. If both values are on the same side of the threshold, the update is safe, i.e., it does not violate the threshold query and cannot invalidate the skyline. Otherwise, the site notifies the coordinator and a synchronization process is initiated.

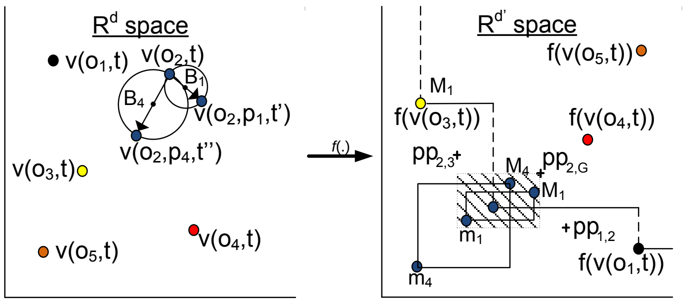


Fig. 4. Handling updates with the pivot-based method: (a) constructing the balls in the $\mathbb{R}^d$ space, (b) constructing the boxes in the $\mathbb{R}^{d'}$ space.

An example is depicted in Fig. 4, with two sites (p_1 and p_4) receiving updates for the same object o_2, and constructing the local bounding balls, B_1 and B_4 (Fig. 4(a)). Let $\overrightarrow{m}_1/\overrightarrow{M}_1$ denote the minimum and maximum values of $\boldsymbol{f}$ inside B_1, as computed at p_1, and $\overrightarrow{m}_4/\overrightarrow{M}_4$ the ones inside B_4. Since both $\overrightarrow{m}_1$ and $\overrightarrow{M}_1$ remain within the safe region defined by the threshold queries in $\mathbb{R}^{d'}$ (Fig. 4(b)), the update at p_1 is safe and registered locally at p_1. On the other hand, the update at p_4 is unsafe, since $\overrightarrow{m}_4$ violates the query corresponding to $\overrightarrow{pp}_{2,3}$. Thus, p_4 notifies the coordinator of its current local vector, initiating a synchronization process.

The local monitoring algorithm also makes use of the more general *safe zone* mechanism for testing local violations (Section II). Safe zones can be defined for various classes of monitoring functions; for instance, using hyperplanes for linear functions. In our work, we employ safe zones whenever applicable, as these can drastically reduce the number of local violations and, consequently, the required network resources. More details on the definition and construction of safe zones can be found in [19], [13].

D. Synchronization

Consider a PIVOT threshold-crossing query Q monitoring the relative dominance relation of the object pair $\{o_i, o_j\}$ that raises a local violation due to an update of object o_i at some site in $\mathcal{P}(o_i)$. As discussed briefly in Section II, the coordinator initiates a *balancing process* to try to resolve the violation on o_i. If that balancing fails to resolve the local threshold violation even after contacting all sites, the coordinator computes the updated $\vec{v}(o_i)$ out of the collected local statistics. Then, if the dominance relation between o_i and o_j has not changed, the coordinator only needs to recompute the pivot point for Q, and send it to $\mathcal{P}(o_i)$ and $\mathcal{P}(o_j)$. Otherwise, it updates the skyline according to the updated global statistics (using a centralized continuous skyline algorithm to reduce computation cost [8]), and recomputes *only* the threshold queries involving at least one of the two objects and a skyline object, according to the process described in Section III-B. All updated and new threshold queries are then sent to the sites monitoring the corresponding objects, and the monitoring protocol continues. The above process relies on cached global statistics vectors of some objects (i.e., o_j), to extract the new threshold queries. It is therefore possible that the local statistics vectors at some of the sites cause immediate threshold violations with the updated threshold queries. In such cases, synchronization is invoked recursively, until no more threshold violations are observed.

An important optimization here is *lazy query updating*, which postpones the replacement of all queries that are still valid, even if the participating objects have changed their skyline status. For example, when an object is removed from the skyline but still dominates a large number of objects, the coordinator need not update the corresponding query. Instead, sites continue monitoring the query, until an update causes a threshold crossing. In our experiments with real-world data sets, this optimization has been shown to enable substantial network savings.

A slight modification is required at the synchronization process for the DIRECT algorithm: Since DIRECT threshold queries are defined on pairs of objects, balancing is always performed for both objects. The rest of the synchronization scheme remains the same.

IV. THE ADAPTIVE METHOD

The geometric method (and, in effect, the proposed algorithms) relies on the existence of a small slack (i.e., freedom to move) for each object, for effectively filtering local updates. In extreme situations, however, the constructed threshold queries may be too tight, leaving little slack for updates and causing frequent synchronizations (e.g., when two objects are very close in $\mathbb{R}^{d'}$). Depending on the frequency and cost of these synchronizations, it may be more network-efficient to identify such costly threshold queries, and exclude their corresponding objects from the geometric monitoring protocol. All updates for these objects are then *directly streamed* to the coordinator, thereby introducing a cost for sending the updates, but eliminating the need for costly synchronizations.

In this section, we propose an adaptive module for identifying such objects. The module is executed by the coordinator each time any object causes a threshold violation, and operates by estimating and comparing the communication cost for keeping the object under geometric monitoring versus directly streaming all its updates. Note that this module is only applicable to PIVOT; since DIRECT always considers objects in pairs, the dependencies across objects make it impossible to exclude an individual object from geometric monitoring.

With $\mathcal{A}_{gm}$ and $\mathcal{A}_{st}$ we denote the two alternative monitoring schemes, the first based on the geometric method (i.e., PIVOT) and the second based on streaming updates. We distinguish two types of threshold violations: (a) *true threshold violations*, where the global statistics vector of the object has changed sufficiently to cause a threshold violation in the global query; and, (b) *false-positive threshold violations*, where only a local statistics vector of the object causes a violation that can be resolved with balancing, without changing the threshold query. Note that both $\mathcal{A}_{gm}$ and $\mathcal{A}_{st}$ will incur the same true threshold violations for the same stream, but only $\mathcal{A}_{gm}$ will run into false-positive violations.

To decide between $\mathcal{A}_{gm}$ and $\mathcal{A}_{st}$ for a given object o, the coordinator needs to predict the network cost required by each scheme for monitoring o until the next true threshold violation for o. Let t denote the time of the last global synchronization for o, and t' the time of the next true threshold violation caused by o. For illustration purposes only, assume that the coordinator has full knowledge of the updates arriving between t and t'. Let $N_{t'}$ denote the number of updates arriving for o in this time range, $N_{fp}(o)$ the number of false positive threshold violations, and $C_{fp}(o)$ the average cost of resolving each such violation. Then, the cost for monitoring o with $\mathcal{A}_{gm}$ is $\mathcal{C}_{gm} = C_{fp}(o) \times N_{fp}(o)$ (for resolving all false positive threshold violations), whereas the cost for $\mathcal{A}_{st}$ is simply $\mathcal{C}_{st} = c \times N_{t'}$, where c is the cost of a single

Algorithm 1: Adaptivity Estimation Algorithm

```
// Executed at the coordinator
function Estimate$N_{t'}$()
begin
    n ← 1;
    TC ← false; // true when I find a threshold crossing
    repeat
        TC ← probe(n); // check for threshold crossing
        if (!TC) then n ← 2n;
    until (TC);
    // I know that $n/2 < N_{t'} \leq n$
    maxN ← n;   minN ← n/2;
    while (maxN-minN>1) do
        n = minN + (maxN-minN)/2;
        if (probe(n)) then maxN ← n; else minN ← n;
    end
    return n;
end

// Checks for threshold crossing, for a given n
function probe(int n)
begin
    for (dim = 1 → d) do
        // Compute left/right bounds for prob 0.5 (see Eqn.3)
        $l[dim]$ ← computeLeftBound($n, 0.5$);
        $r[dim]$ ← computeRightBound($n, 0.5$);
    end
    // sampleN determines the sampling resolution
    for (int sample=0 → sampleN) do
        $\vec{p}$ ←UniformSampleFromHyperCube($l, r$);
        // Compute prob to reach $\vec{p}$ after $n$ steps (see Eqn.3)
        $pr_p$ ←probToReachPoint($\vec{p}, \vec{v}(o,t), n$);
        if ($pr_p \geq 0.5$ and $\boldsymbol{f}(\vec{p})$ causes threshold
        crossing) then return true;
    end
    return false;
end
```

update message. The coordinator chooses the algorithm with the smallest network cost, and notifies the sites monitoring o to switch to that algorithm.

A. Estimating Threshold Violation Costs

Clearly, in a real-world situation, at time $t < t'$, the coordinator cannot know the accurate values of $N_{fp}(o)$, $C_{fp}(o)$, and $N_{t'}(o)$, since these concern future updates in the stream. It can, however, estimate these values through extrapolation on recently observed updates for o. In the remainder of this section, we first describe mathematical models for obtaining these estimates, and then present the detailed algorithm that exploits these models to predict the cost of the geometric and streaming schemes.

Mathematical Preliminaries. To estimate the resolution cost $C_{fp}(o)$, the coordinator employs the average cost for resolving false positive threshold violations over the last ℓ observed violations, where ℓ is a small number, e.g., 10. Estimating $N_{t'}$ and N_{fp} requires a prediction model for future object updates. In the absence of knowledge on the distribution characterizing the updates, we employ a *random walk model* to capture the behavior of object updates. Precisely, the changes in both the global and local statistics vectors for each object o are modeled as d-dimensional random walks. The step length for these walks is determined empirically, by averaging the magnitudes of change for all observed updates of o across all sites.

Let vector $\vec{s}(o)$ denote the average of change magnitudes for the updates observed by all sites in $\mathcal{P}(o)$. According to the random walk model [20], the global statistics vector of o follows a d-dimensional binomial distribution, with variance $\sigma_g[i]^2 = \vec{s}(o)[i]^2 \sum_{p\in\mathcal{P}(o)} n_p$, where n_p denotes the number of updates received for object o at site p since time t. A similar random walk is used to model the local statistics vector of o at each site $p \in \mathcal{P}(o)$: To simplify computation, rather than using per-site update statistics, our model employs the single aggregate change vector $|\mathcal{P}(o)| \times \vec{s}(o)$ for all sites in $\mathcal{P}(o)$ (recall that the global statistics vector is the *average* of the $|\mathcal{P}(o)|$ local statistics vectors). Then, the probability distribution describing the local statistics vector of object o at p is a d-dimensional binomial distribution with variance $\sigma_l[i]^2 = (|\mathcal{P}(o)| \times \vec{s}(o)[i])^2 n_p$ [20].

Through one-sided Chebyshev inequalities we can probabilistically bound the location of the global and local statistics vectors of each object, after n_p updates. Precisely, for any dimension i and any point $l < \vec{v}(o,t)[i]$, the probability of $\vec{v}(o,t')[i]$ crossing l along dimension i is $Pr[\vec{v}(o,t')[i] < l] \leq \frac{\sigma_g[i]^2}{\sigma_g[i]^2 + (\vec{v}(o,t)[i]-l)^2}$. Therefore, the value of l satisfying $Pr[\vec{v}(o,t')[i] < l] > pr$ for a desired minimum probability pr is:

$$l \geq \vec{v}(o,t)[i] - \sigma_g[i]\sqrt{(1-pr)/pr} \qquad (2)$$

Similar inequalities hold for $Pr[\vec{v}(o,t')[i] > r]$ for all $r > \vec{v}(o,t)[i]$, as well as for the probability of a local statistics vector dimension being less than l or greater than r.

Estimation Algorithm. Alg. 1 exploits the above-derived probabilistic inequalities to estimate $N_{t'}(o)$ and N_{fp}. Starting from $n = 1$ and using a combination of doubling and binary search, we find the maximum number of steps n, such that any point $\vec{p}$ reachable from $\vec{v}(o,t)$ with probability higher than 0.5, does not cause a threshold violation. Formally, let $\mathcal{V}_n = \{\vec{p}_1, \vec{p}_2, \dots\}$ denote the (possibly infinite) set of points, such that any $\vec{p} \in \mathcal{V}_n$ satisfies the following condition after n updates, for all dimensions $i = 1, \dots, d$:

$$\prod_{i=1}^{d} pr_i \geq 0.5, \text{with } pr_i = \begin{cases} Pr[\vec{v}(o,t')[i] < \vec{p}[i]], & \text{if } \vec{p}[i] < \vec{v}(o,t)[i] \\ Pr[\vec{v}(o,t')[i] > \vec{p}[i]], & \text{if } \vec{p}[i] > \vec{v}(o,t)[i] \end{cases}$$

The significance of $\mathcal{V}_n$ is that each of the points in the set is likely to be reached from $\vec{v}(o,t)$ after n updates, i.e., with probability ≥ 0.5. $N_{t'}(o)$ is set to the maximum value n, such that for all points $\vec{p} \in \mathcal{V}_n$, $\boldsymbol{f}(\vec{p})$ does not cause a threshold violation for any of the threshold queries for object o. To test the above constraints efficiently, the points $\vec{p}$ are uniformly sampled (using a superimposed grid) over the range defined by l and r, as these are computed per dimension for probability 0.5, (e.g., using Equation 2). The number of repetitions required to estimate $N_{t'}(o)$, is logarithmic in $N_{t'}(o)$, and linear in the resolution of the grid.

The same process is used to predict the number of steps for the next false positive threshold violation, required for estimating the total number of false positive threshold violations N_{fp}. Then, using the described formulas for $\mathcal{C}_{gm}$ and $\mathcal{C}_{st}$, we compute the expected cost for $\mathcal{A}_{gm}$ and $\mathcal{A}_{st}$ and select the most efficient monitoring scheme.

Due to sampling and extrapolation, the above process may fail to detect some local or global threshold violations. A sudden change in stream characteristics may also result in an overestimate or underestimate of the values of N_{fp} or $N_{t'}$. Such inaccuracies, however, do not introduce errors in the skyline; the only possible negative consequence is that the adaptive module selects a suboptimal monitoring algorithm for an object, thereby increasing the monitoring cost.

V. Experimental evaluation

Our experiments were focused on evaluating the network efficiency and scalability of Pivot and Direct, as well as on providing guidelines for selecting the best algorithm for each configuration. Network efficiency was measured in number of messages and transfer volume. Since both algorithms guarantee maintaining the exact skyline, their accuracy was always 100% and is therefore not presented in the results.

As a baseline, we have used the only available alternative for continuous fragmented functional skylines, which streams the updates to a central node (only the updates that actually alter the local statistics vector of an object were considered). In the following, the baseline will be denoted as Centr, due to its central nature. Unless noted differently, the results for Pivot and Direct correspond to the fully-fledged variants of the algorithms, i.e., with query reduction, grouping, and the adaptivity extension. In the vast majority of the experiments, each of these extensions was shown to improve the performance of the algorithms, typically reducing the communication overhead by a factor of two.

Data sets. We have used two publicly available real-world data sets, a massive weather-related data set (denoted with Weather), and the Movielens movie ratings data set (Movies). Furthermore, a set of massive synthetic data streams generated with Kossmann's data generator [1] – the standard generator for evaluation of skyline algorithms – allowed us to study the behavior of the algorithms under different data characteristics. Since Kossmann's generator creates only static data sets, updates were simulated by randomly selecting a site p_i and an object o_j at each step, and shifting the local value of the object to a value uniformly selected within the range $[(1-\text{maxCh})\vec{v}(o_j,p_i,t),\ (1+\text{maxCh})\vec{v}(o_j,p_i,t)]$, with maxCh denoting the *maximum relative change* chosen for the experiment. Unless otherwise specified, the reported results correspond to the average cost over 40 executions, with streams of 10 million updates.

Monitored functions. The proposed algorithms were evaluated using both linear and non-linear functions. For linear functions, we will report results for the identity function of the *average object values*, i.e., $\boldsymbol{f}(\vec{v}(o,t)) = \vec{v}(o,t)$, which enables us to directly observe the influence of the data characteristics to the performance of the algorithms. For non-linear functions, we considered three frequently used functions, variance of a dimension across all sites, euclidean norm on two dimensions, and L2 distance on four dimensions.

Table I summarizes the configuration parameters varied in our experiments, and the default values for each parameter. To avoid repetition, in our discussion we will be noting only the parameters with values different from the default values.

Data sets	
Name	**synthetic**, Weather, Movies
Correlation of dim.	**Independent**, Correlated, Anti-correlated
Max. relative change	0.01, **0.02**, 0.04, 0.08, 0.16
# objects	257, 1000, **2000**, 3000, 4000, 5000, 10681
Experimental Configuration	
Function	**Linear**, Norm, L2 distance, Variance
Dimensions	2, 3, 4, 5
# sites	200, 500, **1000**, 1500, 2000, 2500, 5423

TABLE I. Experimental parameters (default value is bold).

A. Influence of the data characteristics

We first investigate the influence of the following data characteristics to the performance of the proposed algorithms:
- **Correlation between dimensions:** *correlated* (e.g., price Vs performance for computers), *anti-correlated* (price Vs mileage for used cars), or *independent* (shipping cost Vs item price).
- **Maximum change:** We consider values from 1% to 16%.

For this first set of experiments, we have generated different synthetic streams of 2000 two-dimensional objects, varying the properties described earlier. The network was configured such that all objects were monitored by all sites. In order to maintain the stream properties also in the skyline space, $\boldsymbol{f}[0]$ and $\boldsymbol{f}[1]$ were set to be the identity functions on the two dimensions of the objects. The total cost of Centr in these experiments was always 10 million messages totaling 305 Mbytes.

Correlation between dimensions. Fig. 5(a) plots the transfer volume required by Pivot and Direct, as measured at regular stream intervals. Notice that, for illustration purposes, Y axis is interrupted at $y = 0.0065$. Clearly, both Pivot and Direct enable substantial savings for all data sets. In particular, both algorithms require two to three orders of magnitude less transfer volume compared to Centr on the data sets with independent and correlated dimensions. The data set with anti-correlated dimensions is more challenging for the two algorithms, since, due to this anti-correlation, skyline objects end up to be close to each other leading to frequent skyline updates. Nevertheless, even for this data set, both Pivot and Direct still enable around 70% reduction of the network cost compared to Centr. Similar observations are derived by the comparison of the three algorithms in terms of number of messages (Fig. 5(b)).

Also note that Direct is more efficient than Pivot for the streams with the correlated and independent dimensions, both with respect to number of messages and transfer volume. This is not the case for the anti-correlated data set, where Pivot substantially outperforms Direct in terms of number of messages. The reason for this discrepancy is the adaptivity extension of Pivot, which, for the anti-correlated data set, sets a small set of objects (less than 10%) to the streaming algorithm, reducing the threshold crossings and the incurred network cost. The effect of the additivity extension is more visible at the latter part of the stream, since the extension relies on stream statistics to detect the candidate objects. In terms of transfer volume, this difference becomes apparent only at the end of the stream, since the messages exchanged by Pivot also include the pivot point coordinates, and are therefore larger than the messages sent by Direct.

The initialization phase of Pivot and Direct induces a small network cost, for broadcasting the initial threshold queries to all sites. Notice that this is a *one-time cost*, and, therefore, with a small significance for continuous skyline queries. In the previous experiments, the maximum initialization cost over all runs and for both algorithms was found to be less than 25 Kbytes per node, i.e., less than 25 Mbytes total. The total transfer volume (including initialization cost) required by the algorithms is shown in Fig. 5(c) (the figure corresponding to number of messages is almost identical to Fig. 5(b), and is omitted). We see that, for the anti-correlated data set, Centr appears to be more efficient at the early stages of the stream compared to Pivot and Direct. This is expected, since Centr does not require initialization. How-

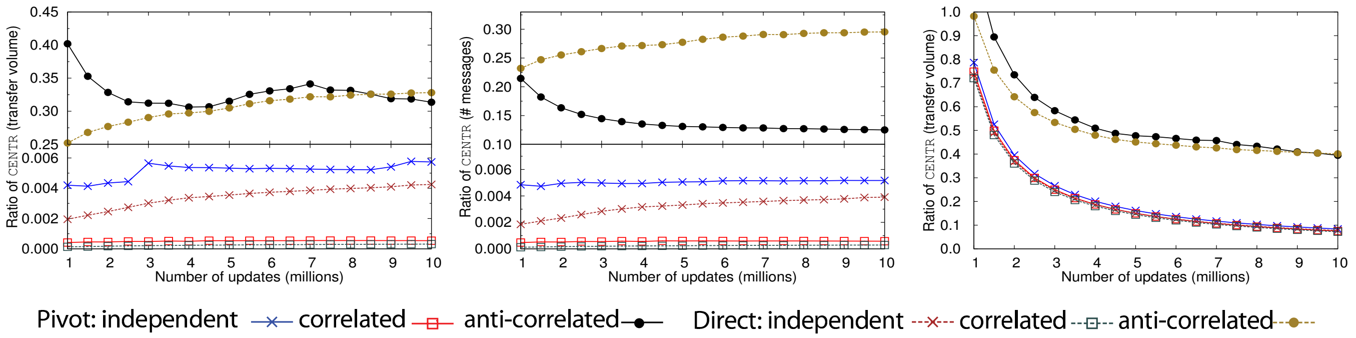


Fig. 5. Effect of the correlation of dimensions to the performance of Pivot and Direct: (a) transfer volume, (b) # messages, (c) transfer volume including one-time initialization cost.

ever, already after around 1.5 million updates, the amortized transfer volume of both Pivot and Direct becomes less than the corresponding cost of Centr. Since most real-world applications involve long-running – possibly infinite – streams, the one-time initialization cost of the proposed algorithms is not an important concern. Instead, as more updates arrive in the stream, the amortized transfer volume of the two proposed algorithms converges to their running transfer volume. Therefore, the running cost of each algorithm (Fig. 5(a) and (b)) is a more interesting evaluation indicator.

Maximum Change. The streams in the previous simulations were generated assuming a maximum relative change maxCh $= 0.02$ per update. This value is reasonable for simulating real-world applications, since the stream readings usually arrive at regular intervals, e.g., every 30 seconds, and therefore most updates are expected to be small. However, to verify the applicability of Pivot and Direct for fast-changing streams, we have also conducted experiments with different maximum change values, up to 0.16. Fig. 6(a) plots the measured transfer volume and number of messages required by Pivot and Direct for data sets generated with independent dimensions. As expected, increasing maxCh results to an increase of the network cost of both algorithms. Nevertheless, even for maxCh $= 0.16$, Pivot enables network savings of 80% compared to Centr in terms of transfer volume, and 90% in terms of messages. Direct is even more efficient, requiring 88% less network volume, and 90% less messages compared to Centr. The 8% difference in the transfer volume between Pivot and Direct is attributed to the more compact threshold queries of Direct, which do not need to include the pivot points. For small maxCh values, the network savings of both algorithms are substantially higher, approximating 100%.

B. *Scalability*

To investigate the scalability of the two algorithms, we also ran experiments with different network sizes. To ensure that each site receives a substantial number of updates for each object, the number of rounds in each experiment was set such that each site receives an expected number of 10000 updates. Therefore, the cost of Centr varied with the network size, starting from 152 Mbytes for 500 sites, and reaching to 763 Mbytes for the largest network of 2500 sites.

The cost of Pivot and Direct for the different network sizes is presented in Fig. 6(b), as a ratio of the corresponding cost of Centr for the same setup. Clearly, both Pivot and Direct maintain a steady cost ratio compared to Centr, independent of the network size (the small peaks visible in the plot for networks of 500 and 1500 sites are due to random artifacts in the generated data sets). For all network sizes, the transfer volume is less than 3% of Centr for Pivot and less than 1% for Direct, whereas the number of messages remains always below 1% for both.

We have also considered experiments with different numbers of objects (from 1000 to 5000). Similar to the previous experiment, the stream size was adapted to the number of objects (5000 expected updates per object, reaching to a total of 25 million updates for the 5000-objects configuration). As seen in Fig. 6(c), the cost ratio for Pivot in terms of transfer volume slightly increases with the number of objects. This behavior is expected, since the denser area around the skyline (attributed to the increased number of objects) leads to more frequent threshold crossings and updates in the skyline. This does not affect the number of messages, because all threshold crossings observed due to an update by a node are packed to a single message. Nevertheless, even for the 5000-objects experiment, the transfer volume of both Pivot and Direct does not exceed 4% of Centr, whereas the number of messages remains always less than 2%. The scalability experiments were also repeated in configurations where each site monitored a subset of the objects, with very similar results.

C. *Different function types*

The final set of experiments with synthetic data focused on investigating the influence of the number of functions to the performance of Pivot and Direct, and on verifying the applicability of the algorithms to different function types – not necessarily linear. Fig. 7(a) presents the performance of Pivot and Direct when monitoring 2, 3, and 4 linear functions, i.e., the skyline space is of 2, 3, and 4 dimensions. Notice that the transfer volume for the baseline varies with the number of functions, since the number of object dimensions are increased. For 2 dimensions, the transfer volume of Centr is 305 Mbytes, for 3 dimensions it is 343 Mbytes, and for 4 dimensions it reaches to 381 Mbytes.

We observe that an increase of the number of functions leads to higher network requirements for both Pivot and Direct. The main reason for this observation is that by adding functions – dimensions in the skyline space – we increase the frequency of synchronizations (recall that threshold crossing in a single dimension is sufficient to invoke the synchronization process). Nevertheless, even for the experiment with 4 functions, Pivot is substantially more efficient than Centr, reducing the transfer volume by 70%, and the messages by more than 80% by the end of the stream. Notice that skylines of higher dimensions are rarely considered, since in high dimensions most of the objects end up in the skyline, rendering it useless. Also note that Pivot is more efficient than Direct for 3 or more functions. The inefficiency of Direct

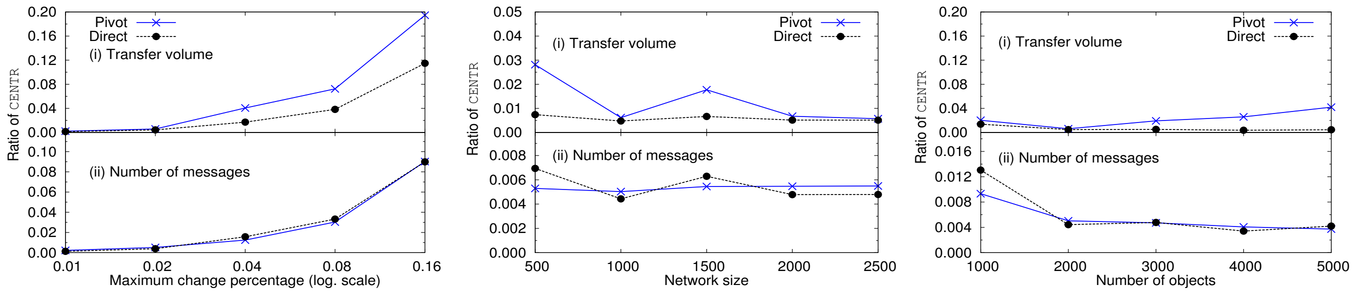


Fig. 6. Effect of (a) maximum relative change, (b) network size, (c) number of objects, to the transfer volume of PIVOT and DIRECT.

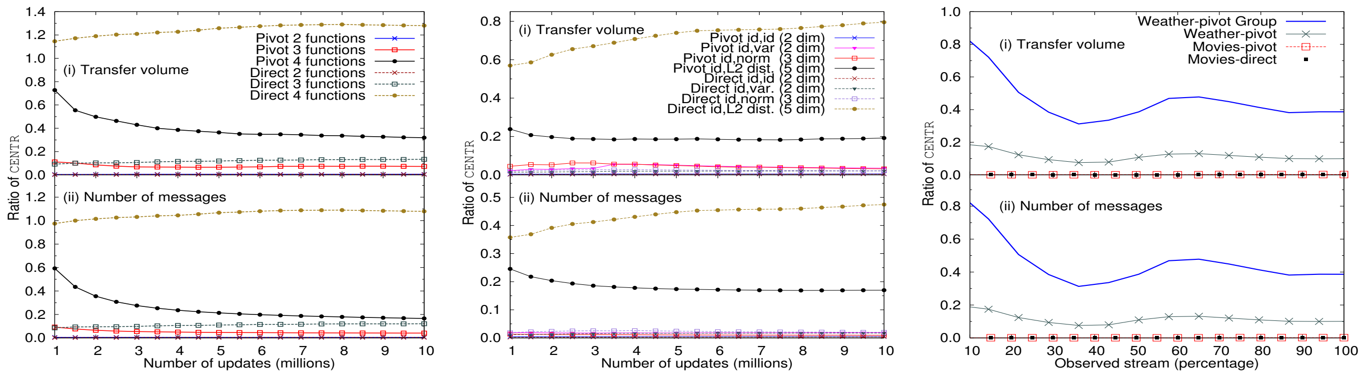


Fig. 7. (a) Effect of the number of functions, (b) Effect of the function types, (c) Experiments with real data sets.

in the experiments with 4 functions is attributed to a high frequency of threshold crossings, due to the increased number of functions. PIVOT on the other hand identifies the objects causing frequent threshold crossings, through the adaptivity extension, and sets them to the streaming algorithm, thereby avoiding the majority of the threshold crossings.

We also conducted experiments with more complex functions, namely the Euclidean norm on two dimensions ($Norm(\vec{v}(o,t)) = \sqrt{\sum_{i=1}^{2} \vec{v}(o,t)[i]^2}$), the L2 distance on four dimensions ($L2(\vec{v}(o,t)) = \sqrt{\sum_{i=1}^{2} \vec{v}(o,t)[i]^2 - \vec{v}(o,t)[i+2]^2}$), and the variance of one dimension on all sites ($Var(\vec{v}(o,t)[i]) = \sum_{p \in \mathcal{P}(o)} (\vec{v}(o,t,p)[i])^2 - \vec{v}(o,t)[i]^2$). In all experiments, the skyline space was 2-dimensional, with $\boldsymbol{f}[0]$ set as the identity function, and $\boldsymbol{f}[1]$ set as one of the three functions above.

Fig. 7(b) plots the network cost incurred by the two algorithms for each function, as the ratio of the corresponding cost of CENTR. For comparison, the figure also includes the cost for the case where both functions are set to the identity function. Notice that, as with the previous experiments, the transfer volume of CENTR was not the same for all functions, since the number of dimensions in the object space differed: for the variance, the transfer volume was 305 Mbytes (2-dimensional objects), for the Euclidean norm 343 Mbytes (3-dim.), and for L2 distance 420 Mbytes (5-dim.).

We see that the proposed algorithms substantially outperform CENTR, also on skylines defined through non-linear functions. The improvement is in fact similar to the improvement observed with linear functions. The only exception involves the experiments with DIRECT used for monitoring the pair of identity and L2 distance functions. For this configuration, DIRECT reduces the transfer volume only by 20% compared to CENTR. DIRECT does not perform well in this configuration due to its local monitoring process, which requires constructing balls in the $2d$-space for each function, i.e., in the 8-dimensions for L2 (cf. Section III-A). This substantially increases the frequency of threshold crossings, and consequently also the transfer volume. PIVOT, on the other hand, reduces the network cost to around 20% of the baseline, since: (a) it constructs balls in the d-dimensional space, and not in the $2d$-dimensional space and, (b) it uses the adaptivity extension, which avoids a large number of threshold crossings.

D. *Experiments with real data sets*

We have also conducted experiments with two real-world data sets, WEATHER and MOVIES. WEATHER was downloaded from the website of the National Oceanic and Atmospheric Administration (NOAA). The data set includes weather statistics collected from a network of sensors distributed around the globe. For our experiments, we used a subset of the data set for years 2010 and 2011, by excluding the sensors with incomplete location meta-data or infrequent readings. The resulting data set contained 93.6 million readings of 5423 sensors distributed in 257 countries. An interesting characteristic of this data set is that, even though the value of each object (country) is fragmented over many sensors, each sensor always maintains the data of *a single object*, i.e., the weather statistics corresponding to a single country. This has two important consequences. First, as shown in Theorem 1, DIRECT is provably worse than PIVOT for such a setup, and therefore we do not use it in this experiment. Second, our experiments have shown that the query grouping extension introduced at Section III-B is not beneficial for this extreme scenario, since every time a small threshold violation occurs at a composite pivot point, a large number of distinct sensors need to be contacted for updating the composite pivot point. Therefore, for this data set, we present results of PIVOT, both with and without query grouping.

MOVIES is the largest of the Movielens data sets, published by the grouplens group. The data set contains 10 million ratings of 10681 movies provided by 71567 users, and is frequently used for evaluating recommender systems. In the context of this work, MOVIES is used to simulate the scenario where a large number of servers distributed around the world (such as eBay servers) collaborate to maintain a set of useful skylines on collected user ratings. Since the data set does not contain any kind of user demographics that would allow us to break the stream to sites, we introduced a random distribution of the users to 200 sites. Each site accepts ratings for all movies, and the initial ratings at each site are set based on a sample of the ratings for the movie.

Fig. 7(c) shows the incurred network cost for maintaining two indicative skylines on these data sets: (1) for WEATHER, the skyline of countries with lower average temperatures and lower average dew points, and, (2) for MOVIES, the movies with the highest average ratings and the highest number of ratings in the network. The transfer volume is always reported as a percentage of the corresponding cost of CENTR, which was 2.8 Gbytes for WEATHER, and 248 Mbytes for MOVIES.

Both methods enable substantial improvement on the incurred network cost, similar to the improvement with the synthetic data sets with different correlations (cf. Fig. 5). With respect to WEATHER, PIVOT without query grouping is more efficient than the fully-fledged PIVOT, requiring 4 times less network cost. Compared to CENTR, PIVOT without grouping requires only 10% of the cost of CENTR, both with respect to number of messages and transfer volume. The network savings for MOVIES approached 100% for both algorithms.

We also see that WEATHER is more difficult to handle compared to MOVIES, i.e., the network savings are lower. This is due to the characteristics of the two data sets. On the one hand, MOVIES has correlated dimensions, i.e., a movie with high average rating is highly likely to have a high number of ratings. As discussed in Section V-A, our algorithms thrive in these data sets, requiring a near-zero network cost. On the other hand, WEATHER has two properties that make it a difficult data set: (a) the similar weather statistics observed in nearby countries, leading to tight threshold queries, and to frequent changes in the skyline, and, (b) the periodicity of the readings due to the day-night cycle, which causes frequent changes in the skyline. Extreme weather situations, such as the extremely low temperatures in continental Europe in the winter of 2010-2011 (starting at around 50% of the stream), also cause drastic skyline changes and increased network requirements. Nevertheless, even with this data set, the overall network savings are significant, reaching to 90%.

Summary. The experimental evaluation showed that the proposed algorithms substantially outperform CENTR, the only available alternative. Cost reduction was frequently in the range of two orders of magnitude, as shown with experiments on both real and synthetic data sets, and using different number and types of functions. Both PIVOT and DIRECT were shown to scale well with the number of objects, and number of sites. Furthermore, a thorough experimental comparison of the two algorithms was used to reveal the preferred algorithms for each situation:

• PIVOT is the algorithm of choice for monitoring dense skyline spaces, i.e., with anti-correlated dimensions, and with many functions, due to the adaptivity extension which detects tight threshold queries and assigns their corresponding objects to streaming monitoring.

• PIVOT substantially outperforms DIRECT when monitoring skylines that include non-linear functions with a high number of dimensions, e.g., the L2 distance.

• For 2-dimensional skylines with correlated or independent dimensions, DIRECT is more efficient than PIVOT, since it does not introduce fixed pivot points, allowing higher slack to the objects, and more compact threshold queries.

VI. CONCLUSIONS

In this paper we formally introduced the problem of continuous fragmented skyline queries, i.e., skyline queries defined over *aggregate values* of distributed data, possibly through additional complex functions. To address the problem, we proposed two distributed algorithms that rely on geometric monitoring to reduce the number of updates that need to be transmitted by each site to a central node, thereby drastically reducing the total network cost for maintaining the skyline. We have also described an adaptivity module which enables detecting highly volatile data and handling them more efficiently. An extensive experimental evaluation with massive real-world and synthetic datasets demonstrated the scalability of the algorithm, as well as its significantly improved network efficiency compared to the only available baseline algorithm.

REFERENCES

[1] S. Borzsony, D. Kossmann, and K. Stocker, “The skyline operator,” in *ICDE*, 2001.

[2] K. Hose and A. Vlachou, “A survey of skyline processing in highly distributed environments,” *VLDB J.*, 2011.

[3] Z. Zhang, R. Cheng, D. Papadias, and A. Tung, “Minimizing the communication cost for continuous skyline maintenance,” in *SIGMOD*, 2009.

[4] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis, “Efficient routing of subspace skyline queries over highly distributed data,” *TKDE*, vol. 22, no. 12, 2010.

[5] W.-T. Balke, U. Gntzer, and J. X. Zheng, “Efficient distributed skylining for web information systems,” in *EDBT*, 2004.

[6] G. Trimponias, I. Bartolini, D. Papadias, and Y. Yang, “Skyline processing on distributed vertical decompositions,” *TKDE*, vol. 25, no. 4, 2013.

[7] I. Sharfman, A. Schuster, and D. Keren, “A geometric approach to monitoring threshold functions over distributed data streams,” in *SIGMOD*, 2006.

[8] P. Wu, D. Agrawal, Ö. Egecioglu, and A. El Abbadi, “Deltasky: Optimal maintenance of skyline deletions without exclusive dominance region generation,” in *ICDE*, 2007.

[9] D. Papadias, G. Fu, M. Chase, and B. Seeger, “Progressive skyline computation in database systems,” *TODS*, vol. 30, no. 1, 2005.

[10] Z. Huang, H. Lu, B. C. Ooi, and A. K. H. Tung, “Continuous skyline queries for moving objects,” *TKDE*, vol. 18, no. 12, 2006.

[11] Y. Tao, X. Xiao, and J. Pei, “Subsky: Efficient computation of skylines in subspaces,” in *ICDE*, 2006.

[12] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou, “Parallel distributed processing of constrained skyline queries by filtering,” in *ICDE*, 2008.

[13] D. Keren, I. Sharfman, A. Schuster, and A. Livne, “Shape sensitive geometric monitoring,” *TKDE*, vol. 24, no. 8, 2012.

[14] G. Cormode and M. Garofalakis, “Approximate continuous querying over distributed streams,” *TODS*, vol. 33, no. 2, 2008.

[15] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi, “Holistic aggregates in a networked world: Distributed tracking of approximate quantiles,” in *SIGMOD*, 2005.

[16] C. Olston, J. Jiang, and J. Widom, “Adaptive filters for continuous queries over distributed data streams,” in *SIGMOD*, 2003.

[17] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk, “Gigascope: A stream database for network applications,” in *SIGMOD*, 2003.

[18] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, “The design of an acquisitional query processor for sensor networks,” in *SIGMOD*, 2003.

[19] S. Burdakis and A. Deligiannakis, “Detecting outliers in sensor networks using the geometric approach,” in *ICDE*, 2012.

[20] R. Graham, D. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 1989.