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

Monitoring Distributed Fragmented Skylines

Odysseas Papapetrou · Minos Garofalakis

the date of receipt and acceptance should be inserted later

Abstract Distributed skyline computation is important for a wide range of 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: 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 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 distribution and object fragmentation. We present the first known distributed algorithms for continuous monitoring of skylines over *complex functions* of *fragmented* multi-dimensional objects. Our algorithms rely on decomposition of the skyline monitoring problem to a select set of distributed threshold-crossing queries, which can be monitored locally at each site. We propose several optimizations, including: (a) a technique for adaptively determining the most efficient monitoring strategy for each object, (b) an approximate monitoring technique, and (c) a strategy that reduces communication overhead by grouping together threshold-crossing queries. Furthermore, we discuss how our proposed algorithms can be used to adress other continuous query types. A thorough experimental study with synthetic and real-

This is a post-peer-review, pre-copyedit version of an article published in Distributed and Parallel Databases. The final authenticated version is available online at http://doi.org/10.1007/s10619-018-7223-7

O. Papapetrou
Data-Intensive Applications and Systems Laboratory, EPFL
E-mail: odysseas.papapetrou@epfl.ch Tel.: +41 21 69 36652

M. Garofalakis
ECE department, Technical University of Crete, Greece
E-mail: minos@softnet.tuc.gr Tel.: +30 28210 37211

life data sets verifies the effectiveness of our schemes and demonstrates order-of-magnitude improvements in communication costs compared to the only alternative centralized solution.

1 Introduction

Since the introduction of the skyline operator [3], the problem of efficiently constructing skylines in distributed environments has been widely studied. 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 skyline 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 wireless sensor networks (where multiple remote sensor measurements must be monitored and analyzed for trends, patterns, intrusions, or other adverse events) and 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). 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.

Continuous skyline maintenance has been addressed in recent work, both for centralized [28,21,14] and distributed deployments [31]. Still, that work, as well as all existing work in distributed skyline processing assumes, at most, *horizontal* or *vertical partitioning* of the data: each site maintains a subset of the complete object vectors [27], or a subset of the dimensions for all objects [2,26]. 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. Thus, each site can independently apply local filtering techniques on the observed updates, drastically reducing the required network resources. Unfortunately, this assumption is unrealistic for a number of real-world, distributed monitoring applications, where the vector of each object is determined *by aggregating* (e.g., averaging) partial vector values fragmented over many sites.

To make matters worse, skyline dimensions can often 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 variance of packet sizes routed to each subnet through each of the edge routers. Such complex *functional* skyline queries are 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 local object position [23].

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

router	target ip	#packets	vol.
1	121.11.*.*	134	1226
1	117.23.*.*	60	72
2	121.11.*.*	180	1280
2	117.23.*.*	80	100
3	121.11.*.*	160	1301
...	...	...	...

target ip	#packets	vol.
121.11.*.*	158	1269
117.23.*.*	70	86
201.7.*.*	627	4874
72.11.*.*	884	982
...	...	...

Aggregation (average)

target ip	avg(#packets)	var(vol.)	sky
121.11.*.*	158	1497	YES
117.23.*.*	70	392	NO
201.7.*.*	627	0	NO
72.11.*.*	884	1208	YES
...	...	...	...

Skyline space: avg(#packets), var(vol.)

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

tools for network administrators, for quickly identifying 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 focus on IPs potentially under attack. Skyline dimensions can even be defined through complex, non-linear functions on the aggregated data, such as the variance of the edge routers' workload per IP (Fig. 1(c)) – a key indicator for sites under 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 challenging due to the volume and volatility of the traffic update streams. Problem is aggravated by the usage of non-linear functions for the definition of the skyline dimensions, in which case a router observing a local update cannot even predict the direction of the change at the skyline space. This calls for a distributed solution for skyline maintenance, where each edge-router can react only to its local updates that potentially invalidate the existing skyline, notifying the central monitor for further analysis. □

Our Contributions. All previous distributed skyline techniques assume either horizontal or vertical partitioning of the data, 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 (and, much more general) setting 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 aggregating (e.g., averaging) the object's (partial) vectors across all sites, and, (b) the skyline space is defined through potentially complex functions, parameterized by the aggregate object values. Our contributions are summarized as follows:

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

• We present two algorithms for efficient processing of continuous fragmented skyline queries, with dimensions defined through arbitrarily complex functions over the aggregate vectors. Our 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. We show how these queries can be monitored efficiently using ideas from the geometric method [15,23].

• We propose several optimizations that significantly improve the communication efficiency of our fragmented skyline monitoring algorithms. These include several techniques for effectively reducing the number of queries (which can result in substantial communication gains), an approximation technique for error-tolerant setups,

and a technique based on random-walk models for adaptively determining the most efficient monitoring strategy for each object.

• We discuss how PIVOT and DIRECT can be exploited and adapted for monitoring other types of continuous queries over fragmented data.

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

2 Related Work

Since the introduction of the skyline operator [3], several aspects of skyline computation have been explored, such as, continuous skylines, e.g., [28,21,14], functional (or, *dynamic*) skylines [21] subspace skylines [25], and skylines over distributed and P2P networks [13]. 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 studied in recent years (see [13] 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., [27]; (2) *Point Filtering:* Representative points, belonging to one or more sites' local skylines, can help other sites effectively reduce their local skylines [2,26]; and, (3) *Site Filtering:* Compact local site summaries can be used to target neighboring sites that can potentially contribute skyline points [9]. However, at the core of all these approaches is the requirement that the value of each dimension for each object is always maintained by a single site; that is, objects are vertically or horizontally partitioned, but not fragmented (as the IP data in our example above). Even though both vertical and horizontal partitioning hold significant interest for real-life applications (and, in fact, can also be handled by our work), our contributions lie in devising the *first known schemes* for the general 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. [31] for distributed continuous skyline monitoring, 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 that of threshold-crossing queries employed 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, the algorithm of [31] and our algorithms (without the adaptivity extension) produce similar types of local constraints, yet, each following different optimization strategies. Note, however, that [31] supports neither fragmented data nor functional skylines, the combination of which is the main focus of our work. Some ideas from [31], 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.

Cheema et al. [4] recently proposed a centralized skyline monitoring algorithm for moving skyline queries based on "safe zones". Even though we also utilize (a different notion of) "safe zones" in this work, we focus solely on distributed environments, and address the challenges that arise due to data fragmentation. As such, the way we define, construct, and exploit safe zones is completly unrelated to [4]. Instead, our techniques build on ideas from distributed geometric monitoring [15, 23].

To reduce pairwise comparisons, existing works rely on grouping and representing two or more objects with a single representative, i.e., a pivot point [18, 30]. This approach bears similarity to one of our algorithms (PIVOT), which relies on pivot points to represent large space regions. However, PIVOT chooses and utilizes pivot points in a completely different manner. First, pivot points in our setting are not actual object points; they are the cleverly-chosen points in the high-dimensional space. The way of computing these pivot points (where to place them, and which objects to represent with them) is a core contribution of our work, and makes substantial difference in terms of performance. Second, all the machinery of our algorithm focuses on reducing the network cost, and not the computational overhead, in contrast to the previous works.

Summarizing, none of the existing techniques handles, or can be easily extended to address the problem considered in this work. The difficulty stems mainly from the fact that in our setting, sites are not aware of the global object values – each site only knows its own local values of each object. This hinders the additivity property, which constitutes the core of most previous algorithms, and disables effective point and site filtering. An additional difficulty comes from the fact that we are considering a streaming setup. Therefore, the skyline in our setup needs to be continuously maintained, as it can be invalidated after each update.

An early version of this work has been presented in [22]. Compared to [22], this article includes several novel contributions, including: (a) a new grouping technique for threshold-crossing queries that can reduce network cost by *up to a factor of two* (Section 5), (b) an extension of the ideas to enable approximate skyline monitoring that can further reduce network in domains where small (bounded) inaccuracies are acceptable (Section 4.5) (c) a discussion on how the proposed algorithms can be used for addressing other continuous query types, e.g., skyband queries, constrained skylines, and skylines over uncertain data (Section 7), and (d) a more extensive experimental evaluation (Section 8).

3 Preliminaries

Problem Formulation. 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., [7,20,6,1,10], our distributed stream-processing model does not allow direct communication between remote sites; instead, remote sites exchange messages only with the coordinator, providing it with state information on its (locally-observed) streams. Such a hierarchical processing model is 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* $\mathbf{v}(o_j, p_i, t)$. The global state of o_j is defined as the average (or, more generally, any convex combination) of o_j's local statistics vectors across all sites in $\mathcal{P}(o_j)$, i.e., the *global statistics vector* $\mathbf{v}(o_j, t) = \frac{1}{|\mathcal{P}(o_j)|} \sum_{p_i \in \mathcal{P}(o_j)} \mathbf{v}(o_j, p_i, t)$. (To simplify notation, we omit the explicit time dependence 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 skyline dimensions are defined through a d'-dimensional *function vector* $\boldsymbol{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$, where each dimension $\boldsymbol{f}[k](\mathbf{v}(\cdot))$ is a possibly complex function over the original d-dimensional global statistics vectors of the objects. We define the notion of *functional dominance* (or, $\boldsymbol{f}$*-dominance*) over fragmented data objects as follows. (Wlog., in this article we assume that lower values are preferred.)

Definition 1 ($\boldsymbol{f}$-dominance) Let $\mathbf{v}(o_i)$, $\mathbf{v}(o_j)$ 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 $\boldsymbol{f}[k](\mathbf{v}(o_i)) \leq \boldsymbol{f}[k](\mathbf{v}(o_j))$ for all $k \in \{1, \ldots, d'\}$, and $\exists k \in \{1, \ldots, d'\}$ such that $\boldsymbol{f}[k](\mathbf{v}(o_i)) < \boldsymbol{f}[k](\mathbf{v}(o_j))$.

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 $\mathcal{O}$.

Example 2. Building on the ISP monitoring scenario of Example 1, the set of remote processing sites $\mathcal{P}$ includes all edge routers of the ISP, 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 the $\boldsymbol{f}$*-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 $\mathbf{v}(o_i, p_j)$ for each IP address o_i: $\mathbf{v}[0](o_i, p_j)$ stores the count of all observed packets destined for o_i and routed through p_j, $\mathbf{v}[1](o_i, p_j)$ stores the sum of the packet sizes, and $\mathbf{v}[2](o_i, p_j)$ stores $(\mathbf{v}[1](o_i, p_j))^2$. The global statistics vector for each IP address o_i is the average of the local statistics vectors over all routers, i.e., $\mathbf{v}(o_i) = \sum_{p_j \in \mathcal{P}(o_i)} \mathbf{v}(o_i, p_j)/|\mathcal{P}(o_i)|$. The desired skyline space is then defined by function $\boldsymbol{f}$: $\boldsymbol{f}[0] = \mathbf{v}[0](o_i)$, i.e., the identity function of the average number of packets for each IP address, and $\boldsymbol{f}[1] = Var(\{\mathbf{v}(o_i, p_j) | p_j \in \mathcal{P}(o_i)\}) = \sum_{p_j \in \mathcal{P}(o_i)} \frac{\mathbf{v}[2](o_i, p_j)}{|\mathcal{P}(o_i)|} - \left(\sum_{p_j \in \mathcal{P}(o_i)} \frac{\mathbf{v}[1](o_i, p_j)}{|\mathcal{P}(o_i)|}\right)^2 = \mathbf{v}[2](o_i) - (\mathbf{v}[1](o_i))^2$. $\square$

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 are huge [8]), or power and bandwidth restrictions (e.g., in wireless sensor networks, where communication overhead is the key factor determining sensor battery life [19]). 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 or fail to react to important transitions in a timely manner. 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.

3.1 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 required elements of the geometric method. Further details can be found in [23].

The geometric method addresses the basic problem of monitoring *distributed threshold-crossing queries*; that is, monitor whether $f(\mathbf{v}(o)) < \tau$ or $f(\mathbf{v}(o)) > \tau$, for any arbitrary, possibly complex, non-linear function $f()$ of a global statistics vector $\mathbf{v}(o)$ 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 $f(\mathbf{v}(o))$, 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 $\mathbf{v}(o, p, t_0)$ to a coordinator, where the global statistics vector $\mathbf{v}(o, t_0)$ is computed. This global statistics vector is also called the global estimate vector $\mathbf{e}(o)$, 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 $\Delta\mathbf{v}(o, p_j) = \mathbf{v}(o, p_j, t) - \mathbf{v}(o, p_j, t_0)$. The *drift vector* is then defined as $\mathbf{u}(o, p_j) = \mathbf{e}(o) + \Delta\mathbf{v}(o, p_j)$. These vectors can be used to bound the location of the global statistics vector, which is guaranteed to lie within the convex hull formed by the drift vectors of all nodes and $\mathbf{e}(o)$ [23]. Therefore, by checking that the convex hull does not overlap the inadmissible region (i.e., the region $\{\mathbf{v} \in \mathbb{R}^2 : f(\mathbf{v}) > \tau\}$ 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

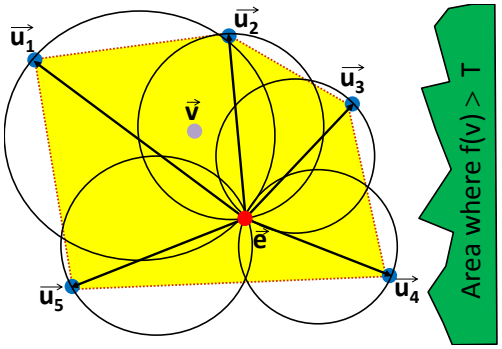

Fig. 2 Estimate vector $\mathbf{e}$, delta vectors $\Delta\mathbf{v}(p_i)$ (arrows out of $\mathbf{e}$), convex hull enclosing the current global vector $\mathbf{v}$ (dotted outline), and bounding balls $B(\mathbf{e}, \Delta\mathbf{v}(p_i))$.

the global condition into a local constraint, we place a d-dimensional *bounding ball* around each local delta vector, of radius $||\mathbf{e}(o) - \mathbf{u}(o, p_j)||/2$ and centered at $(\mathbf{e}(o) + \mathbf{u}(o, p_j))/2$ (see Fig. 2). It can be shown that the union of all these balls completely covers the convex hull of the drift vectors [23]. Therefore, as long as the bounding ball constructed individually at each node is *monochromatic*, i.e., it does not overlap with the inadmissible 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 $\Delta\mathbf{v}(p_i)$ 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 $\Delta\mathbf{v}(p_i)$. This process involves collecting the current delta vectors from a subset of the sites, and recomputing the minimum and maximum values of $f(\mathbf{v})$ 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. [15] 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.

4 Monitoring Fragmented Skylines

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. Their main difference lies in the details of this decomposition into queries. Still, since both algorithms share a common framework, we describe them in parallel, with references to their particularities.

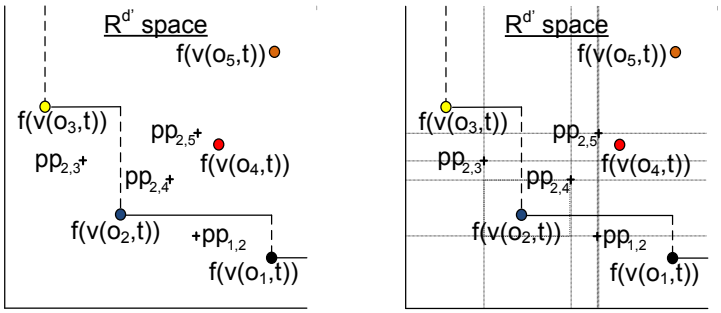

Fig. 3 Pivot-based method: (a) the four pivot points for o_2 in the $\mathbb{R}^{d'}$ space, (b) the safe region for o_2.

We start with a high-level description of the distributed-monitoring protocol. Initially, the user configures the continuous skyline query, by defining the (possibly complex) functions over the global statistics vector $\mathbf{v}$ to derive the skyline dimensions. 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 any standard, centralized algorithm. Then, for each object $o_i \in \mathcal{O}$, the coordinator extracts a set of 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 guarantee skyline correctness: *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 in Section 3.1, a threshold-crossing query focuses on detecting when the value of a function $g()$ over a fragmented dynamic vector crosses a threshold value τ. Let t_0 denote the query construction time and let $\mathbf{v}(t)$ be the dynamic vector. Then, using the sign function $\mathrm{sgn}()$, we can define this general threshold-crossing query $Q_{t_0}(g, \mathbf{v}, \tau)$ as the boolean condition:

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

The above boolean condition will become true at any time t only if there is a threshold crossing of $g(\mathbf{v}(t))$, i.e., $g(\mathbf{v}(t)) > \tau$ and $g(\mathbf{v}(t_0)) < \tau$, or vice-versa. Clearly, both $g()$ and τ can be multi-dimensional, giving rise to a query that is equivalent to the OR of the boolean conditions across all dimensions. To keep our descriptions concise, we employ the multi-dimensional form of Query (1) over our skyline function vector $\boldsymbol{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$. 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. Then, we outline the local monitoring and synchronization processes, which are largely common to both algorithms.

4.1 Threshold-Crossing Query Decomposition

We now discuss the details of decomposing a continuous fragmented skyline into threshold-crossing queries. 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.

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 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, $\overrightarrow{pp}_{i,j} = \frac{1}{2}(\boldsymbol{f}(\mathbf{v}(o_i, t_0)) + \boldsymbol{f}(\mathbf{v}(o_j, t_0)))$. Then, it constructs the two threshold-crossing queries: $Q_{t_0}(\boldsymbol{f}, \mathbf{v}(o_i), \overrightarrow{pp}_{i,j})$ (installed at sites $\mathcal{P}(o_i)$) and $Q_{t_0}(\boldsymbol{f}, \mathbf{v}(o_j), \overrightarrow{pp}_{i,j})$ (installed at sites $\mathcal{P}(o_j)$). As an example, Fig. 3(a) depicts a sample data set with five 2-dimensional points in the $\boldsymbol{f}$-skyline space, including the four pivot points defined for o_2 with respect to all other objects. Any site observing o_2 then has to monitor the following threshold-crossing queries (one per pivot point): $Q_{t_0}(\boldsymbol{f}, \mathbf{v}(o_2), \overrightarrow{pp}_{1,2})$, $Q_{t_0}(\boldsymbol{f}, \mathbf{v}(o_2), \overrightarrow{pp}_{2,3})$, $Q_{t_0}(\boldsymbol{f}, \mathbf{v}(o_2), \overrightarrow{pp}_{2,4})$, and $Q_{t_0}(\boldsymbol{f}, \mathbf{v}(o_2), \overrightarrow{pp}_{2,5})$.

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 per dimension $k = \{1, \ldots, d'\}$, namely, $\{\mathbf{x} : \mathbf{x}[k] < \overrightarrow{pp}_{i,j}[k]\}$, $\{\mathbf{x} : \mathbf{x}[k] > \overrightarrow{pp}_{i,j}[k]\}$, and $\{\mathbf{x} : \mathbf{x}[k] = \overrightarrow{pp}_{i,j}[k]\}$. For each dimension k, $\boldsymbol{f}(\mathbf{v}(o_i))$ belongs in exactly one of these subspaces. The intersection of these d' subspaces containing $\boldsymbol{f}(\mathbf{v}(o_i))$ across all threshold-crossing queries for object o_i effectively defines a *safe region* for o_i; that is, as long as $\boldsymbol{f}(\mathbf{v}(o_i))$ 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(b) depicts the (shaded) safe region for o_2. The threshold-crossing queries installed at $\mathcal{P}(o_i)$ monitor exactly this safe-region condition for o_i. It is not difficult to se that this scheme is correct: as long as no threshold-crossing query fires for any object, 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 the same subset of remote sites, i.e., $\mathcal{P}(o_i) = \mathcal{P}(o_j)$. We define the function-difference vector $\boldsymbol{g}(\mathbf{v}(o_i)|\mathbf{v}(o_j)) = \boldsymbol{f}(\mathbf{v}(o_i)) - \boldsymbol{f}(\mathbf{v}(o_j))$, where $\mathbf{v}(o_i)|\mathbf{v}(o_j)$ 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 $Q_{t_0}(\boldsymbol{g}, \mathbf{v}(o_i)|\mathbf{v}(o_j), \mathbf{0})$

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 ($\mathbf{0}$ denotes the all-zero d'-dimensional vector). For instance, in the example of Fig. 3, the set of DIRECT threshold queries extracted for o_2 is $\mathcal{Q}(o_2) = \{Q_{t_0}(\boldsymbol{g}, \mathbf{v}(o_2)|\mathbf{v}(o_j), \mathbf{0}) : j = 1, 3, 4, 5\}$. Again, it is easy to see that the skyline does not change as long as none of the threshold-crossing queries fire.

A number of issues with DIRECT are worth noting. First, 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 to higher communication costs. (This issue can be avoided for certain function types, e.g., when $\boldsymbol{f}$ is linear, but not in 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 sites (i.e., $\mathcal{P}(o_i) \neq \mathcal{P}(o_j)$), and its effectiveness in such settings. To ensure correctness, the threshold query over $\mathbf{v}(o_i)|\mathbf{v}(o_j)$ 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). To keep the average correct, we need to scale each of the local statistics vectors of o_i 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 balls, thereby increasing the number of local violations. Theorem 1 formalizes this observation, in comparison to PIVOT.

Theorem 1 *Monitoring the* DIRECT *threshold-crossing query* $Q_{t_0}(\boldsymbol{g}, \mathbf{v}(o_i)|\mathbf{v}(o_j), \mathbf{0})$ *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* $Q_{t_0}(\boldsymbol{f}, \mathbf{v}(o_i), \overrightarrow{pp}_{i,j})$, *when all functions in* $\boldsymbol{f}$ *are linear, and* $r = \frac{|\mathcal{S}|}{|\mathcal{P}(o_i)|} > 2$.

Proof We will use Q^p to denote the threshold-crossing query between objects o_i and o_j monitored by PIVOT, and Q^d the query monitored by DIRECT. We will show that when both queries are instantiated with the same data, i.e., with identical object values at time t_0, the minimum required update $\mathbf{u}_d$ of o_i that will cause a threshold crossing on Q^d is smaller than the corresponding minimum required update $\mathbf{u}_p$ for Q^p. Therefore, Q^d will be violated more frequently, causing more synchronizations. For simplicity, we examine only the case for a function vector $\boldsymbol{f}$ where all constituting functions are linear, and we focus only on object o_i, i.e., we consider o_j to be stationary on the node receiving the update of o_i. This can happen, e.g., when the node p monitoring o_i does not monitor o_j, or when it did not receive any update for o_j since the last synchronization.

Consider any node $p \in \mathcal{P}(o_i)$ receiving an update $\mathbf{u}$ for o_i at time t. This update will cause a threshold crossing for Q^p only if $\mathrm{sgn}(\boldsymbol{f}(\mathbf{v}(o_i, t)) - \tau) \neq \mathrm{sgn}(\boldsymbol{f}(\mathbf{v}(o_i, t_0)) - \tau)$, with $\tau = (\boldsymbol{f}(\mathbf{v}(o_i, t_0)) + \boldsymbol{f}(\mathbf{v}(o_j, t_0)))/2$. Since $\boldsymbol{f}$ is linear, $\boldsymbol{f}(\mathbf{v}(o_i, t)) = \boldsymbol{f}(\mathbf{v}(o_i, t_0) + \mathbf{u}) = \boldsymbol{f}(\mathbf{v}(o_i, t_0)) + \boldsymbol{f}(\mathbf{u})$.

Recall that $\boldsymbol{f}$ is a function vector. We need to consider each dimension k of $\boldsymbol{f}$ separately. A threshold crossing due to dimension k will occur when $\mathrm{sgn}(\boldsymbol{f}(\mathbf{v}(o_i, t_0))[k] + \boldsymbol{f}(\mathbf{u})[k] - \tau[k]) \neq \mathrm{sgn}(\boldsymbol{f}(\mathbf{v}(o_i, t_0))[k] - \tau[k])$. Without loss of generality, assume that $\boldsymbol{f}(\mathbf{v}(o_i, t_0))[k] < \boldsymbol{f}(\mathbf{v}(o_j, t_0))[k]$ (the other case is symmetric). Then, $\tau[k] >$

$\boldsymbol{f}(\mathbf{v}(o_i,t_0))[k]$, and threshold crossing on Q^p can occur only when $\boldsymbol{f}(\mathbf{u})[k]$ surpasses $\tau[k] - \boldsymbol{f}(\mathbf{v}(o_i,t_0))[k]$, i.e., $\boldsymbol{f}(\mathbf{u}_p)[k] > \frac{\boldsymbol{f}(\mathbf{v}(o_j,t_0))[k] - \boldsymbol{f}(\mathbf{v}(o_i,t_0))[k]}{2}$.

Now consider the case of DIRECT. Q^d will be violated in dimension k when

$$\begin{aligned}\mathrm{sgn}(\boldsymbol{f}(\mathbf{v}(o_i,t))[k] - \boldsymbol{f}(\mathbf{v}(o_j,t))[k]) \neq \\ \mathrm{sgn}(\boldsymbol{f}(\mathbf{v}(o_i,t_0))[k] - \boldsymbol{f}(\mathbf{v}(o_j,t_0))[k])\end{aligned} \quad (2)$$

By our assumption that $\boldsymbol{f}(\mathbf{v}(o_i,t_0))[k] < \boldsymbol{f}(\mathbf{v}(o_j,t_0))[k]$, we know that $\mathrm{sgn}(\boldsymbol{f}(\mathbf{v}(o_i,t_0))[k] - \boldsymbol{f}(\mathbf{v}(o_j,t_0))[k]) = -1$. Therefore, a threshold crossing will be caused only when the LHS of ineq. 2 becomes positive:

$$\begin{aligned}\mathrm{sgn}(\boldsymbol{f}(\mathbf{v}(o_i,t))[k] - \boldsymbol{f}(\mathbf{v}(o_j,t))[k]) = +1 \Rightarrow \\ \boldsymbol{f}(\mathbf{v}(o_i,t))[k] - \boldsymbol{f}(\mathbf{v}(o_j,t))[k] > 0\end{aligned} \quad (3)$$

As discussed in the paper, to account for the fact that o_i is not monitored by all nodes, we need to scale the local statistics drift vector for o_i by $r = |\mathcal{S}|/|\mathcal{P}(o_i)|$. Since $\boldsymbol{f}$ is linear, $\boldsymbol{f}(\mathbf{v}(o_i,t)) = \boldsymbol{f}(\mathbf{v}(o_i,t_0) + r\mathbf{u}) = \boldsymbol{f}(\mathbf{v}(o_i,t_0)) + r\boldsymbol{f}(\mathbf{u})$. Substituting $\boldsymbol{f}$ in Eqn. 3, and since $\mathbf{v}(o_j,t) = \mathbf{v}(o_j,t_0)$, we get $\boldsymbol{f}(\mathbf{v}(o_i,t_0))[k] + r\boldsymbol{f}(\mathbf{u})[k] - \boldsymbol{f}(\mathbf{v}(o_j,t_0))[k] > 0$. Therefore, the condition for threshold crossing becomes $\boldsymbol{f}(\mathbf{u}_d)[k] > \frac{\boldsymbol{f}(\mathbf{v}(o_i,t_0))[k] - \boldsymbol{f}(\mathbf{v}(o_j,t_0))[k]}{r}$. Thus, if $r > 2$, for all dimensions k, $\boldsymbol{f}(\mathbf{u}_d)[k]$ will be smaller than $\boldsymbol{f}(\mathbf{u}_p)[k]$, which directly implies that Q^d will be violated with a smaller magnitude update. □

4.2 Elimination of Redundant Queries

Both algorithms maintain the skyline by monitoring the pairwise dominance between all objects in $\mathcal{O}$. For this, they require $O(|\mathcal{O}|^2)$ queries. However, not all changes in pairwise dominance relations between objects in $\mathcal{O}$ are necessary. For example, the skyline will not change if o_4 (Fig. 3(a)) is updated such that it no longer $\boldsymbol{f}$-dominates o_5, since both o_4 and o_5 continue to be dominated by o_2. In fact, only two types of threshold-crossing queries can signify a change in the skyline: (1) Queries monitoring the domination of a non-skyline object by a skyline object and (2) Queries monitoring the pareto optimality of a skyline object. All other queries are 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. In the example of Fig. 3(a), this gives rise to threshold queries for the pairs $\{o_2, o_4\}$ and $\{o_2, o_5\}$.

(2) Queries Monitoring Pareto-optimality of a Skyline Object: A skyline object o_i may exit the skyline only when another 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.) However, 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(a) 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 $\Theta(|\mathcal{O}|^2)$ to (at most) $2(|\mathcal{O}| + s(d' - 1))$, where s denotes the size of the skyline (and, typically, $s << |\mathcal{O}|$). This set of threshold queries is *sufficient* and *minimal* for accurate fragmented-skyline monitoring.

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

Proof We will prove that the threshold queries are sufficient for detecting whenever an object changes status, i.e., enters or leaves the skyline. The proof is valid for both PIVOT and DIRECT. First, we consider the simpler case of an object o_i not belonging in the skyline at time t_0, to show that it cannot enter the skyline without first causing a threshold violation. For o_i, the algorithm constructs a threshold crossing query between o_i and an object o_j that dominates o_i. As long as the threshold query is not violated by an update of either o_i or o_j, o_j continues to dominate o_i, which guarantees that o_i does not enter the skyline.

Second, we consider an object o_i that belongs in the skyline at time t_0. We will prove that o_i cannot be removed from the skyline without first causing a threshold violation, which will enable the coordinator to detect the change in the skyline. o_i can be removed from the skyline only due to an update of o_i or an update of any object o_j, which will dominate o_i. We have the following cases:

• o_j, which did not belong in the skyline at time t_0, is updated and dominates o_i. Since o_j dominates an object that was previously skyline object, this means that o_j first needs to become part of the skyline. This, of course, corresponds to the case addressed earlier, thus causing a violation of the thresold query that monitors o_j and enabling the coordinator to detect the skyline update.

• Object o_j, which belonged in the skyline at time t_0, is updated and now dominates o_i. Since at time t_0 object o_j did not dominate o_i, there existed at least one dimension k for which $\boldsymbol{f}(\mathbf{v}(o_i, t_0))[k] < \boldsymbol{f}(\mathbf{v}(o_j, t_0))[k]$. Also, since o_j also belonged in the skyline, our monitoring algorithm constructed a threshold query between o_j and its immediate skyline neighbor o_h at dimension k that satisfies $\boldsymbol{f}(\mathbf{v}(o_h, t_0))[k] < \boldsymbol{f}(\mathbf{v}(o_j, t_0))[k]$. There are two cases: (a) o_h is the object o_i, in which case the corresponding threshold query will be violated, or (b) o_h is not o_i, in which case $\boldsymbol{f}(\mathbf{v}(o_i, t_0))[k] < \boldsymbol{f}(\mathbf{v}(o_h, t_0))[k]$ (by the definition of o_h), and the threshold query of o_j corresponding to o_h will be violated. In both cases, the violation will cause synchronization, which will enable the coordinator to detect the change in the skyline. (Note that o_h will also

be monitoring its nearest dominating neighbor in the skyline (say, o_l) so that, if o_l at some point takes the position of the nearest neighbor of o_j, then o_h would fire; in general, it is not difficult to see that some monitoring rule will fire if the nearest skyline neighbor of o_j changes, so we can assume that the monitored nearest skyline neighbor is always current.)

We also need to prove that all constructed threshold queries are required for correctly monitoring the skyline. Again, we need to consider the two types of queries separately.

• Queries monitoring domination of a non-skyline object: Recall that only one query is constructed. If this query is removed for any non-skyline object o_i, then the algorithm will not be able to track the location of o_i, possibly masking skyline updates.

• Queries monitoring dominance of a skyline object: Two queries are constructed per dimension, with the two immediate skyline neighbors. By removing any of these queries for a skyline object o_i, then we will not be able to track the location of the object in the corresponding dimension, possibly masking skyline updates. □

4.3 Local Monitoring

The threshold-crossing queries produced by the 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 queries without imposing centralization of updates. Briefly, a node receiving an update for an object o forms the bounding ball (see Section 3.1), and tests for monochromicity w.r.t. all threshold queries. This test is performed by finding the minimum and maximum values 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 cannot invalidate the skyline. Otherwise, the site notifies the coordinator to initiate the synchronization process.

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 $\mathbb{R}^{d'}$ (the gray-shaded area in Fig. 4(b)), the update at p_1 is safe and registered locally at p_1. However, 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 3.1). Safe zones can be defined for various classes of monitoring functions, and recent work has shown that they can reduce network cost for monitoring threshold-crossing queries by an order of magnitude [17].

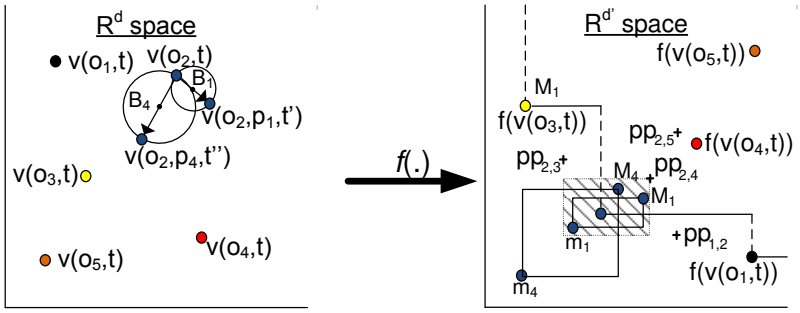

Fig. 4 Handling updates with the pivot-based method: (a) constructing the balls in the $\mathbb{R}^d$ space, (b) constructing the boxes in the $\mathbb{R}^{d'}$ space.

4.4 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 3.1, the coordinator initiates a *balancing process* to resolve the violation on o_i. If the process fails to resolve the local threshold violation even after contacting all sites, the coordinator computes the updated $\mathbf{v}(o_i)$ out of the collected local statistics. Then, if the dominance relation between o_i and o_j has not changed, the coordinator recomputes the pivot point for Q, and sends 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 [28]), 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 4.2. 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 does not update the corresponding query. Instead, sites continue monitoring the query until an update causes a threshold crossing.

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.

4.5 Approximate monitoring

To further reduce network cost, both PIVOT and DIRECT support approximate monitoring. With approximate monitoring, we target applications that can tolerate errors in the skyline, as long as the misclassified objects are very near the skyline border. For example, consider monitoring the skyline of IP addresses with the highest number of packets and transfer volume across all ISP routers. Small errors around the skyline region can often be tolerated; for instance, it is acceptable if an IP address is misclassified as belonging in the skyline, as long as it is very close to the skyline border, i.e., it can become a member of the skyline with a small shift. It is even the case that, due to sampling or sketching (both of which are frequently used in network monitoring), IP statistics may already be approximate. Similar inaccuracies are also introduced in sensor networks, due to hardware limitations of the sensors. As such, small errors are already inherent in many applications, without reducing the importance or utility of skyline queries.

Moving along the lines of previous works (ADRs [16] and skylines of coarser scales [12]), we define approximation quality by bounding the maximum allowed error per object. That is, any misclassified object must be very near the skyline border, i.e., with maximum distance ϵ at each dimension. This is achieved by defining approximate threshold queries (as opposed to standard threshold queries generated by the exact PIVOT and DIRECT algorithms), which are represented as follows:
$Q_{t_0}(g, \mathbf{v}, \tau, \epsilon) \equiv \text{sgn}(g(\mathbf{v}(t_0)) - \tau) \neq \text{sgn}(g(\mathbf{v}(t)) - \tau + \lambda)$ with

$$\lambda = \begin{cases} -\epsilon, & \text{if } g(\mathbf{v}(t_0)) < \tau \\ +\epsilon, & \text{if } g(\mathbf{v}(t_0)) > \tau \end{cases}$$

Intuitively, an approximate threshold query allows a local violation by a maximum of ϵ without initiating the synchronization process. This local violation does not necessarily translate to a skyline update, since in most cases a local violation does not translate to a global threshold violation.

Approximate threshold queries can be utilized by PIVOT and DIRECT as follows. PIVOT guarantees a maximum error ϵ by constructing approximate threshold queries with an acceptable error $\epsilon/2$: since each object can violate the pivot point by at most $\epsilon/2$ (in opposite directions), the dominance relation between two objects will be violated by at most ϵ. DIRECT does not need to pre-allocate this error, since the dominance relation between object pairs is always checked with a threshold query that includes both objects. Therefore, DIRECT constructs approximate queries with error parameter ϵ.

Clearly, there exist alternative expressions of approximate threshold queries, e.g., ϵ could be relative on the pivot point location, or it could be a d'-dimensional vector enabling different accuracy requirements per dimension. The best approach is determined by the application scenario; it is straightforward to adapt PIVOT and DIRECT to handle alternative expressions.

5 The Adaptive Method

The geometric method (and, in effect, the proposed algorithms) relies on the existence of a small slack for each object, for effectively filtering local updates. In extreme situations, the constructed threshold queries may be too tight, leaving no 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 is more network-efficient to identify these few threshold queries, and exclude their corresponding objects from the geometric monitoring protocol. All updates for these objects are *directly streamed* to the coordinator, thereby introducing a cost for sending the updates, but eliminating the need for frequent costly synchronizations.

We now propose an adaptive module for identifying such cases. The module is executed by the coordinator each time an 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 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 changes sufficiently to cause a threshold violation in the query; and, (b) *false-positive threshold violations*, where only a local statistics vector causes a violation that can be resolved with balancing, without changing the threshold query.

To decide between $\mathcal{A}_{gm}$ and $\mathcal{A}_{st}$ for any 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' (we will remove this assumption later). 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 update message, since $\mathcal{A}_{st}$ does not incur false-positive violations. The coordinator chooses the algorithm with the smallest network cost, and notifies the sites monitoring o to switch to that algorithm.

Clearly, the challenge now is to estimate the values of $N_{fp}(o)$, $C_{fp}(o)$ and $N_{t'}(o)$, since these depend on future parts of the stream. The coordinator estimates these values through extrapolation on recently observed updates for o. We now first describe the 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.

5.1 Estimating threshold violation costs

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 κ observed violations, where κ is a small number, e.g., 100. 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 $\mathbf{s}(o)$ denote the average of change magnitudes on the global statistics vector $\mathbf{v}(o)$, for the updates observed by all sites in $\mathcal{P}(o)$. According to the random walk model [11], $\mathbf{v}(o)$ follows a d-dimensional binomial distribution, with variance $\sigma_g[i]^2 = \mathbf{s}(o)[i]^2 \sum_{p \in \mathcal{P}(o)} n_p$, 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 $\mathbf{v}(o,p)$ 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 $|\mathcal{P}(o)| \times \mathbf{s}(o)$ for all sites in $\mathcal{P}(o)$. Then, the probability distribution describing the local statistics vector of object o at p is a d-dimensional binomial distribution with variance $\sigma_l[i]^2 = (|\mathcal{P}(o)| \times \mathbf{s}(o)[i])^2 n_p$.

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

$$l \geq \mathbf{v}(o,t)[i] - \sigma_g[i]\sqrt{(1-pr)/pr} \tag{4}$$

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

Estimation Algorithm. The derived inequalities can be exploited to estimate $N_{t'}(o)$ and N_{fp} (cf. Alg. 1). Starting from number of steps $n = 1$, and using a combination of doubling and binary search (lines 5-13), we find the maximum number of steps n, such that any point $\mathbf{p}$ reachable from $\mathbf{v}(o,t)$ with probability higher than $pr = 0.5$, does not cause a threshold violation. Formally, let $\mathcal{V}_n = \{\mathbf{p}_1, \mathbf{p}_2, \ldots\}$ denote the (possibly infinite) set of points, such that any $\mathbf{p} \in \mathcal{V}_n$ satisfies the following condition after n updates, for all dimensions $i = 1, \ldots, d$:

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

The significance of $\mathcal{V}_n$ is that each of the points in the set is likely to be reached from $\mathbf{v}(o,t)$ after n updates, i.e., with probability ≥ 0.5. $N_{t'}(o)$ is set to the maximum value n, such that for all points $\mathbf{p} \in \mathcal{V}_n$, $\boldsymbol{f}(\mathbf{p})$ does not cause a threshold violation for any of the threshold queries of object o.

Algorithm 1: Adaptivity Estimation Algorithm

```
   // Executed at the coordinator
1  function Estimate N_{t'}()
2  begin
3      n ← 1
4      TC ← false // Set to true when I find a threshold crossing
       // Doubling to determine upper bound
5      repeat
6          TC ← probe(n) // check for threshold crossing
7          if (!TC) then n ← 2n;
8      until (TC);
       // Binary search: I know that n/2 < N_{t'} ≤ n
9      maxN ← n,  minN ← n/2
10     while (maxN-minN>1) do
11         n = minN + (maxN-minN)/2
12         if (probe(n)) then maxN ← n ;
13         else minN ← n ;
14     end
15     return n
16 end
   // Checks for threshold crossing, for a given n
17 function probe(int n)
18 begin
19     for (dim = 1 → d) do
           // Compute left/right bounds for prob 0.5 (see Eqn.2)
20         l[dim] ← computeLeftBound(n, 0.5)
21         r[dim] ← computeRightBound(n, 0.5)
22     end
       // sampleN determines the sampling resolution
23     for (int sample=0 → sampleN) do
24         p ←UniformSampleFromHyperCube(l, r)
           // Compute prob to reach p after n steps (see Eqn.3)
25         pr_p ←probToReachPoint(p, v(o, t), n)
26         if (pr_p ≥ 0.5 and f(p) causes threshold crossing) then return true
27         ;
28     end
29     return false
30 end
```

To test the above constraints efficiently, the points $\mathbf{p}$ are uniformly sampled over the range defined by l and r, as these are computed per dimension for probability 0.5, using Equation 4. This test is performed by function `probe` (lines 16-28). The function first computes the left and right bounds per dimension using Eqn. 4, and then performs uniform sampling (using a superimposed grid) to check if the probability to reach the point after n steps is above the probability threshold pr. In this case, the point is checked for possible threshold crossing. The algorithm returns the smallest value of n for which there exists a point $\mathbf{p}$ reachable from $\mathbf{v}(o,t)$ in n steps with probability higher than pr, that can cause a threshold crossing. 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 thresh-

old violations N_{fp}. 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, this 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.

Special cases. The described algorithm relies on sampling to estimate the frequency of threshold crossings. Sampling can be avoided for certain function types by computing directly the necessary minimal shift in the $\mathbb{R}^d$ space that causes a threshold violation. For instance, for a linear function $\boldsymbol{f}$, the necessary minimal shift in the $\mathbb{R}^d$ space that causes a threshold violation corresponds to the minimum absolute distance per dimension of the current global statistics vector of the object, and the coordinates in the set $\{\boldsymbol{f}^{-1}(\overrightarrow{pp}_1), \boldsymbol{f}^{-1}(\overrightarrow{pp}_2), \ldots\}$, where $\boldsymbol{f}^{-1}$ denotes the inverse function of $\boldsymbol{f}$, and $\overrightarrow{pp}_i$ are used to denote the pivot points constructed for the object. Then, we can employ the probabilistic inequalities (e.g., Equation 4) to estimate the minimal values of $N_{fp}(o)$ and $N_{t'}(o)$ that will lead to threshold violations. The same optimization is applicable to weakly-monotonic non-linear functions, as well as multimodal functions.

6 Grouping Of Threshold Queries

Since all threshold crossing queries need to be propagated and monitored in the network, the number of queries influences the algorithm's network performance. In Section 4.2 we have shown how to eliminate all queries that are not necessary for guaranteeing the correctness of the skyline, reducing the total number from $O(|\mathcal{O}|^2)$ to $O(|\mathcal{O}|)$. However, even after this reduction, skyline objects with dense dominance regions may end up participating in a large number of queries. We now show how to further reduce these queries by grouping. Our discussion first focuses on PIVOT. We discuss grouping for DIRECT in Section 6.3.

PIVOT 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 constructed based on the composite pivot points, which are much fewer than the original ones, and typically enable enlarging the safe regions for the non-skyline objects. The coordinator decides which pivot points should be grouped for each object o_i on the basis of the position of the pivot points (basic grouping), and of the expected maintenance cost for the resulting composite threshold queries (advanced grouping).

6.1 Basic grouping for PIVOT

Any two pivot points $\overrightarrow{pp}_{i,j}$ and $\overrightarrow{pp}_{i,h}$ for an object o_i can be grouped together only if they reside on the same side of $\boldsymbol{f}(\mathbf{v}(o_i))$ in all d' dimensions, i.e., $\text{sgn}(\boldsymbol{f}[k](\mathbf{v}(o_i))-$

$\overrightarrow{pp}_{i,j}[k]) = \text{sgn}(\boldsymbol{f}[k](\mathbf{v}(o_i)) - \overrightarrow{pp}_{i,h}[k])$ for $k = 1, \ldots, d'$. All pivot points belonging in the same group $G = \{\overrightarrow{pp}_{i,j}, \overrightarrow{pp}_{i,h}, \ldots\}$ are then replaced by a composite pivot point $\overrightarrow{pp}_{i,G}$ that imposes equivalent constraints on $\boldsymbol{f}(\mathbf{v}(o_i, t))$. If o_i dominates the pivot points in G, $\overrightarrow{pp}_{i,G}$ takes the minimum value per dimension out of all the pivot points in the group (otherwise the maximum value is taken). Formally, the composite point is defined as follows: $\overrightarrow{pp}_{i,G}[k] = \begin{cases} \min_{\overrightarrow{pp} \in G} \overrightarrow{pp}[k] \text{ if } \min_{\overrightarrow{pp} \in G}(\overrightarrow{pp}[k]) \geq \boldsymbol{f}[k](\mathbf{v}(o_i, t)) \\ \max_{\overrightarrow{pp} \in G} \overrightarrow{pp}[k] \text{ if } \max_{\overrightarrow{pp} \in G}(\overrightarrow{pp}[k]) < \boldsymbol{f}[k](\mathbf{v}(o_i, t)) \end{cases}$

for $k = 1, \ldots, d'$. The pivot points in G for objects $\{o_j, o_h, \ldots\}$ are also replaced by the composite pivot point $\overrightarrow{pp}_{i,G}$, which can enable additional slack for these objects. For example, for object o_2 from Fig. 3(b), basic grouping will replace $G = \{\overrightarrow{pp}_{2,4}, \overrightarrow{pp}_{2,5}\}$ with a single composite pivot point that coincides with $\overrightarrow{pp}_{2,4}$. Basic grouping guarantees that the total number of queries is effectively reduced to (at most) $n + 2sd'$.

Basic grouping performs cost-agnostic grouping, which can be problematic in some cases. We illustrate this limitation with an example. Consider the setup shown in Fig. 5. For simplicity assume that each object is monitored by 2 nodes, and nodes do not overlap, i.e., $\mathcal{P}(o_i) \cap \mathcal{P}(o_j) = \emptyset$, for all objects o_i, o_j. Basic grouping will construct a single query for both non-skyline objects with a pivot point that coincides with $pp_{1,2}$. However, since $pp_{1,2}$ is very close to o_1 and o_2, it will likely be frequently violated. Each true threshold crossing will require updating the pivot point also to the two nodes holding o_3, even though $f(\mathbf{v}(o_3))$ is very far from $f(\mathbf{v}(o_1))$. Instead, if we keep $pp_{1,2}$ and $pp_{1,3}$ ungrouped, we will only need to update the nodes holding o_1 and o_2, thereby saving two messages at every threshold crossing violation. Our next algorithm addresses this limitation.

6.2 Advanced grouping for PIVOT

Advanced grouping method (Alg. 2) avoids this limitation by taking into account both the expected frequency of threshold crossings and the cost of updating the pivot

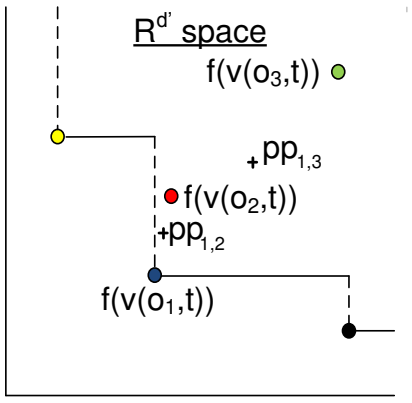

Fig. 5 A problematic scenario for basic grouping.

points in order to decide which pivot points should be grouped together. First, the coordinator executes the basic grouping algorithm and extracts an initial set of groups $\mathcal{G}_i = \{G_{i,1}, G_{i,2}, \ldots\}$ for each skyline item o_i. Each group $G_{i,j} \in \mathcal{G}_i$ is further split to a set of sub-groups $\mathcal{G}'_{i,j} = \{G'_1, G'_2, \ldots\}$, as follows. For each pivot point $\overrightarrow{pp} \in G_{i,j}$, the coordinator simulates the addition of $\overrightarrow{pp}$ in each of the groups of $\mathcal{G}'_{i,j}$ (initially empty), and estimates the increase on the amortized network cost per update of o_i until the next *true* threshold violation of the group's composite threshold query. Also, the expected amortized network cost for adding a new group in $\mathcal{G}'_{i,j}$ that contains only $\overrightarrow{pp}$ is computed. The pivot point is added to the group that minimizes the expected cost increase, or, more formally, to the group $\min\arg_{g \in \{\mathcal{G}'_{i,j} \cup \emptyset\}} C(g \cup \overrightarrow{pp}) - C(g)$, where $C(\cdot)$ denotes the amortized cost of the group.

The challenge in the above process is to estimate the network cost of each potential query group. This cost corresponds to the network resources required for deploying the corresponding composite query to all relevant nodes, plus the resources for handling the false positive threshold violations through balancing. An observation that simplifies our calculations is that the number of false positive threshold violations, and the cost of handling them, are orthogonal to the grouping method – they only depend on the coordinates on the ungrouped pivot points. As such, these can be ignored for our computation, and we can focus on finding the grouping that minimizes the deployment cost only.

Note however that each potential group has a different lifetime. The lifetime of a group (and its corresponding composite query) is defined as the number of updates arriving for the object between the query construction time and its next true threshold violation. To make queries comparable, we amortize the cost of each query over its whole lifetime, which can be estimated using the approach presented in Section 5. In detail, let $\mathcal{P}(G')$ denote the set of sites monitoring at least one of the objects corresponding to the pivot points in G', $Q_{G'}$ the corresponding (candidate) threshold query, and $N_{t'}^{G'}(o_i)$ the estimated number of updates of o_i until the next true threshold violation of $Q_{G'}$ (cf. Section 5). A true threshold violation can only be resolved by updating the composite pivot point at all sites monitoring the threshold query. Therefore, the network cost for the lifetime of a constructed group will be $|\mathcal{P}(G')|$ messages. The same cost amortized per update will be $|\mathcal{P}(G')|/N_{t'}^{G'}(o_i)$. The cost increase due to an addition of a pivot point in G' is computed by subtracting the old amortized cost per update for G' (before adding the candidate pivot point) by the new amortized cost (after adding the new pivot point). The algorithm is sketched in Alg. 2.

Clearly, the discussed greedy algorithm does not always lead to the optimal grouping, since its results depend on the processing order of the pivot points. Errors may also be introduced due to the estimation algorithm for the lifetime of each query. However, these errors do not lead to inaccuracies in the maintained skyline; they may only lead to suboptimal grouping and network performance of the algorithm. As we show later with experiments, advanced grouping performs well in both real-world and synthetic data, addressing the weaknesses of the basic grouping strategy.

Algorithm 2: Advanced grouping

```
   // Executed at the coordinator
   // Set of groups G_i for object o_i is constructed by basic grouping
 1 function groupPivotPoints(G_i)
 2 begin
 3     for (group G_{i,j} ∈ G_i) do
 4         G'_{i,j} ← ∅
 5         for (pivot point p⃗p ∈ G_{i,j}) do
 6             bestGroup ← findBestGroup(G'_{i,j}, p⃗p)
 7             if (bestGroup=NEWGROUP) then
 8                 Construct a new group G' containing p⃗p
 9                 G' ← G' ∪ G'
10             else
11                 Add p⃗p to bestGroup
12             end
13         end
14     end
15     return G'
16 end
   // Find the best group for the pivot point
17 function findBestGroup(G'_{i,j}, p⃗p)
18 begin
19     minDiff ← ∞, bestGroup ← NULL
20     for (Group G' ∈ G'_{i,j}) do
21         N^{G'}_{prev'} ← EstimateN_{t'}()
           // Simulate addition of p⃗p to the group
22         p_new ← new composite pivot point for G' ∪ p⃗p
23         N^{G'}_{t'} ← EstimateN_{t'}()
24         diff = |P(G' ∪ p⃗p)|/N^{G'}_{t'} − |P(G' ∪ p⃗p)|/N^{G'}_{prev'}
25         if (diff<minDiff) then
26             bestGroup ← G'
27             minDiff ← diff
28         end
29     end
       // Simulate creation of a new group for p⃗p
30     p_new ← p⃗p
31     N^{G'}_{t'} ← EstimateN_{t'}()
32     if (|P(p⃗p)|/N^{G'}_{t'} <minDiff) then bestGroup ←NEWGROUP
33     ;
34     return bestGroup
35 end
```

6.3 Grouping for DIRECT

It is straightforward to adapt both basic and advanced grouping for DIRECT: instead of building composite pivot points, we build composite 'dynamic' objects, which get updated with the objects participating in the query. Notice however two important differences compared to PIVOT. First, grouping in DIRECT does not increase the slack of threshold queries for non-skyline objects, since there are no fixed pivot points involved. Second, grouping does not affect the cost of handling threshold violations, since only the invalidated parts of the threshold query need to be replaced. Never-

theless, basic grouping can still be combined with the described representation for computational performance at the monitoring sites.

7 Extensions and additional applications

Besides classic skyline queries, many other interesting skyline variants and application domains have emerged in the last few years, e.g., skycubes, constrained skylines, skybands, and skylines over sliding windows. To illustrate the flexibility provided by PIVOT and DIRECT, we now explain how the two algorithms can be applied to some of these contexts.

Constrained skyline queries. Constrained skyline queries [21] restrict the input domain of a skyline to the objects that satisfy some pre-defined constraints. For example, when booking a hotel room, a traveler may want to set an acceptable price range (between \$100 and \$200) as a hard constraint. The constrained skyline is not necessarily a subset of the full skyline, and therefore it cannot be produced by filtering out the objects of the full skyline according to the constraints. Instead, the objects not satisfying the constraints should be filtered out at the input of the algorithm. This is not straightforward to achieve in the case of data fragmentation, since each node cannot know whether the system-wide average values of any of its objects satisfies the constraints or not.

Both PIVOT and DIRECT can be used to directly track constrained skylines over fragmented data, by computing the skyline over an auxiliary space defined by helper functions. In particular, we only need to define one helper function per dimension that penalizes the tuples not satisfying all constraints. As an example, consider implementing the constraint that the room price is between \$100 and \$200. To get the distance Vs price skyline under this constraint, we compute the skyline over the space produced by the functions $\boldsymbol{f}_{dist}$ and $\boldsymbol{f}_{price}$, defined for pushing the objects not satisfying the constraints outside of the skyline, as follows: $\boldsymbol{f}_{dist}$ = distance if $100 \leq$ price ≤ 200, or $\boldsymbol{f}_{dist} = \infty$ otherwise ($\boldsymbol{f}_{price}$ is defined similarly). The constraints are not necessarily rectangular, as in the described example; any constraint that can be formally described by a function of any form (not necessarily linear) can be integrated in the method.

Monitoring many concurrent skyline queries We frequently require concurrent monitoring of many skyline queries on (partially) overlapping dimensions for the same data input. For instance, a user may need to monitor the skyline of hotel vacancies using price and distance from the beach, whereas another user may be interested on the skyline of hotel vacancies using price and distance from the city center. Concurrent execution of many constrained skyline queries with different constraints (e.g., different price ranges for the hotel rooms) is another example.

Clearly, we can execute multiple independent instances of PIVOT and DIRECT. However, if the skylines to be tracked contain overlapping dimensions, we can save network and computational effort by reducing the number of threshold-crossing queries. In particular, after the coordinator extracts all threshold-crossing queries for each of the desired skylines, it uses the rules described in Section 4.2 to select the tightest threshold-crossing queries, and propagates only these queries to the nodes for

monitoring. Threshold violations and synchronizations are then used for updating all skylines in parallel.

The performance improvement compared to the individual monitoring of each skyline depends on the overlap of the threshold-crossing queries of the different skylines. However, high overlap of threshold-crossing queries is expected in many application scenarios, e.g., when maintaining the skycube [29], or when monitoring many skyline queries with different constraints [21].

Sliding window skylines Algorithms for maintaining skylines over sliding windows have also been proposed, e.g., [24]. However, none of them considers data fragmentation. Both PIVOT and DIRECT can maintain sliding window skylines straightforwardly, by monitoring the auxiliary space defined by a sliding window function, i.e., $\boldsymbol{f}$ will be a sliding window function.

Skyband queries A k-skyband query includes all objects dominated by *at most* k other objects [21] (skyline queries are in fact specializations of skyband queries, with $k = 0$). A key observation towards efficient monitoring of k-skyband queries over fragmented data is that before applying the reduction rules (cf. Section 4.2), both PIVOT and DIRECT will construct sufficient threshold-crossing queries to enable monitoring the dominance relations of all pairs of objects. Therefore, a first approach is to monitor all threshold-crossing queries. As such, the k-skyband query will not change as long as none of these dominance relations is invalidated. To further reduce network and monitoring complexity, we can rewrite the reduction rules such that: (a) each object not belonging in the k-skyband only needs to monitor that it is still dominated by k objects from the skyband and (b) each object o_i from the k-skyband needs to monitor that it does not become dominated by any new object o_j. The number of threshold crossing queries can be reduced further by noticing that, for the second type of queries, it is sufficient to monitor only the pairwise relation of o_i with all objects that are immediate neighbors to at least one dimension (the correctness proof is similar to the one provided for the standard PIVOT and DIRECT algorithms, in Section 4.2). Furthermore, we can use query grouping to reduce the number of threshold crossing queries even more.

Alternative aggregation functions Up to now, we have considered that aggregate values are computed by averaging the object values across several sites. However, any convex combination of the values of each object across the sites can be used (a combination of values is convex if the aggregate value lies within the convex hull of all values). For instance, a weighted average can also be used, where weights may represent, e.g., the trustworthiness or the coverage of the node. Weights that vary with time, and individual weights per dimension are supported by utilizing the ability of PIVOT and DIRECT to define skyline spaces through abritrary functions, as follows: Let vector $\mathbf{w}(o, p_i, t)$ denote the varying weight per node for all dimensions. Then, the skyline dimension is defined by function vector $\boldsymbol{f} = \sum_{p_i \in \mathcal{P}(o)} \mathbf{w}(o, p_i, t) \times \mathbf{v}(o, p_i, t) / \sum_{p_i \in \mathcal{P}(o)} \mathbf{w}(o, p_i, t)$, which can again be monitored with the geometric method.

Operating on uncertain data Uncertain data occurs in many domains, e.g., because of inaccuracies in the data extraction process. Bounding boxes are often utilized in these domains, to represent the area where each uncertain point may lie (see, e.g., [5]). In these cases, it may be required to maintain the skyline of bounding

boxes, i.e., the set of bounding boxes that are not fully dominated by another box. Both PIVOT and DIRECT can be used for monitoring this skyline, albeit with some modifications. This is achieved by constructing threshold-crossing queries that monitor the relative positioning of bounding boxes. In particular, at initialization time we extract the aggregate bounding box per item, and compute the initial skyline of bounding boxes. Then, we extract the threshold crossing queries to monitor the following: a) for each bounding box that belongs in the skyline, we need to monitor that this does not become fully dominated by another bounding box, and b) for the bounding boxes that do not belong in the skyline, we need to monitor that they remain fully dominated by at least one bounding box. These bounding-box threshold crossing queries are essentially reduced to simple threshold-crossing queries between the upper-right coordinate of the dominating bounding box and the lower-left coordinate of the dominated bounding box. As such, the reduction rules and all optimizations proposed in the previous sections can also be used.

8 Experimental evaluation

The experiments were focused on evaluating the network efficiency and scalability of PIVOT and DIRECT, and on providing guidelines for selecting the best algorithm for each configuration. 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 altered 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 otherwise mentioned, PIVOT and DIRECT will be used to denote the full-fledged algorithms, i.e., PIVOT with advanced grouping and adaptive monitoring, and DIRECT with basic grouping.

Data sets. We have used two publicly available real-world data sets, denoted as WEATHER and MOVIES. WEATHER, was downloaded from the National Oceanic Atmospheric Administration website (`http://www.ncdc.noaa.gov/isd`), and includes weather statistics collected in 2010-2011 from a network of 5423 sensors distributed around 257 countries. It contains 93.6 million readings of temperature and dew point (only the updates that altered the local values were considered). MOVIES is the largest of the Movielens data sets (`http://grouplens.org/datasets/movielens/`), containing 10 million ratings of 10681 movies, provided by 71567 users. Since the data set does not contain user demographics, we have introduced a random distribution of users to 200 sites.

Furthermore, a set of massive synthetic data streams generated with the data generator of [3] allowed us to study the behavior of the algorithms under different data characteristics. Since the 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 $[(1-\text{maxRC})\mathbf{v}(o_j,p_i,t), (1+\text{maxRC})\mathbf{v}(o_j,p_i,t)]$, with maxRC denoting the *maximum relative change* chosen for the experiment.

Monitored functions. The algorithms were evaluated using both linear and non-linear functions. For linear functions, we will report results for the identity function

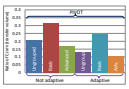

Fig. 6 (a) MOVIES data set, (b) WEATHER data set.

on the *average object values*, i.e., $\boldsymbol{f}(\mathbf{v}(o,t)) = \mathbf{v}(o,t)$, which enables us to directly observe the influence of 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.

Performance indicators. We measured the number of messages and transfer volume required by each algorithm. To enable direct comparison with the baseline, the cost for DIRECT and PIVOT will always be presented as a percentage of the corresponding cost of CENTR on the same setup. We do not report accuracy since both PIVOT and DIRECT offer strict error guarantees, i.e., all errors are always at most equal to the chosen acceptable error ϵ.

Table 1 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.

8.1 Evaluation of algorithmic components

We start by evaluating the proposed algorithm extensions, i.e., query grouping for PIVOT and DIRECT, and adaptive monitoring for PIVOT. We conduct our experiments using the real-world data sets WEATHER and MOVIES. An interesting characteristic of WEATHER 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 of a single country. As shown by Theorem 1, DIRECT is provably worse than PIVOT for such a setup, and is therefore not used on this data set.

Figures 6(a)-(b) present the total network cost (both initialization cost and running cost) induced by all algorithmic variants of PIVOT and DIRECT for maintaining two indicative skylines: (1) for MOVIES, the movies with the highest average ratings and the highest number of ratings in the network, and, (2) for WEATHER, the skyline of countries with lower average temperature and dew point. Costs for PIVOT and DIRECT are reported as a percentage of the corresponding cost of CENTR (248 Mbytes for MOVIES and 2.8 Gbytes for WEATHER).

Our first observation from the MOVIES data set (Fig. 6(a)) is that all algorithmic variants enable substantial network savings compared to CENTR. For this data set, DIRECT is the most efficient algorithm, closely followed by PIVOT with grouping (both basic and advanced grouping). In terms of number of messages (not shown

Data sets	
Name	**synthetic**, WEATHER, MOVIES
Correlation bet. dim.	**Independent**, Correl., Anti-correl.
Max. relative change	0.01, **0.02**, 0.04, 0.08, 0.16
# objects	[257 - 50.000] (default **2000**)
Experimental Configuration	
Function	**Linear**, Norm, L2 dist., Var.
Dimensions	$d' \in \{\mathbf{2}, 3, 4, 5\}$, $d \in \{\mathbf{2}, 3, 4, 5\}$
# sites	[200 - 50.000] (default **1000**)
Acceptable error ϵ	**0**, 0.0005 - 0.05

Table 1 Experimental parameters (default value is bold).

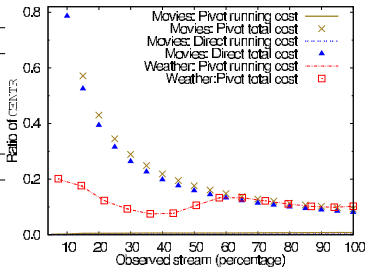

Fig. 7 Total and runnning cost per algorithm

in the figure), all variants require less than 1% of the corresponding messages of CENTR).

As expected, grouping has no noticeable effect on the network performance of DIRECT (cf. Section 6.3). In contrast, grouping for PIVOT not only enables sending of fewer pivot points, but also increases the available slack for non-skyline objects, further reducing the network requirements by a factor of 2. Interestingly, advanced grouping and adaptive monitoring do not provide additional benefits for this data set. This is simply because the skyline stabilizes very early in this data set (after a few thousand updates). As such, there are very few threshold violations, and both optimizations, which focus on reducing the number and cost of threshold violations, have a negligible effect.

Fig. 6(b) plots the required transfer volume for WEATHER. The best performing variant is the full-fledged algorithm (advanced grouping and adaptive monitoring). Interestingly, basic grouping has a negative effect for this setting. This result is due to the cost-agnostic property of basic grouping, which causes problems in scenarios where objects are monitored by subsets of nodes. We also see that adaptive monitoring is beneficial for this data set, further reducing network cost by a factor of two, by setting 3 cities on average on streaming monitoring (cf. Table 2).

The initialization phase of both algorithms induces a small network cost for sending all threshold queries to the participating sites. This is a one-time cost, with a small significance for long-running continuous queries, where running cost is expected to be the dominant factor. For comparison purposes, Fig. 7 presents both total and running cost of each algorithm for the two data sets, as measured at regular stream intervals. For clarity, the plot includes only the transfer volume ratio for the best performing variants of the algorithms, i.e., PIVOT with advanced grouping and adaptive monitoring, and DIRECT with basic grouping. We see that this one-time initialization process raises the total cost ratio of both algorithms at the early stages of the stream. This behavior is particularly visible with the MOVIES data set, which has a larger number of objects that translate to many threshold queries. However, as more updates arrive in the stream, the total transfer volume of the proposed algorithms converges to their running transfer volume. Since most real-world applications involve long-running – possibly infinite – streams, the one-time initialization cost of the algorithms is not an important concern. Instead, the running cost of each algorithm is a more interesting evaluation indicator.

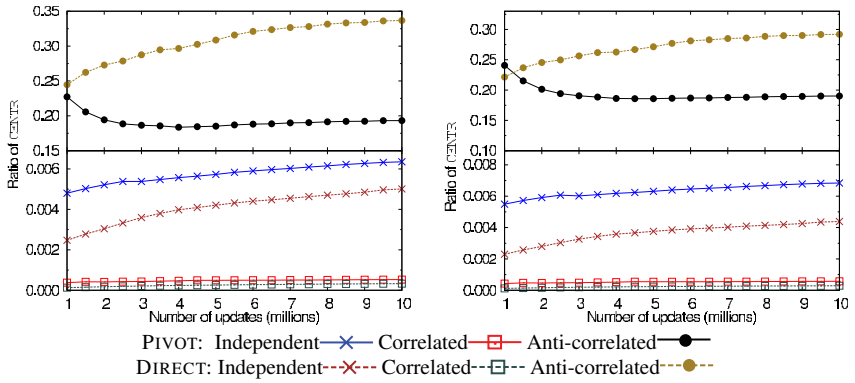

Fig. 8 Effect of dimensions correlation to PIVOT and DIRECT: (a) transfer volume, (b) # messages.

We also see that running cost for the MOVIES data set is close to zero, for both algorithms. This is expected, since MOVIES has a relatively stable skyline. WEATHER on the other hand is more challenging, having a running cost that fluctuates between 5% and 10%. This is attributed to two properties of the 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, the overall network savings are significant, reaching 90% by the end of the stream.

Summarizing, the first set of experiments has shown that both PIVOT and DIRECT substantially outperform CENTR. In the remainder of this section, we will be focusing on the best variants of the two algorithms, i.e., PIVOT with advanced grouping and adaptive monitoring, and DIRECT with basic grouping.

8.2 Influence of data characteristics

We now resort to synthetic data sets in order to investigate the influence of the following data characteristics to the performance of the compared 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%.

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 at 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. Figures 8(a)-(b) present the running cost of PIVOT and DIRECT for data sets with different correlations, as measured at regular stream

Data Set	# streaming objects
MOVIES	0.07
WEATHER	3.03
Independent	5
Correlated	0.35
Anti-correlated	138.5

Table 2 Average number of streaming objects in PIVOT.

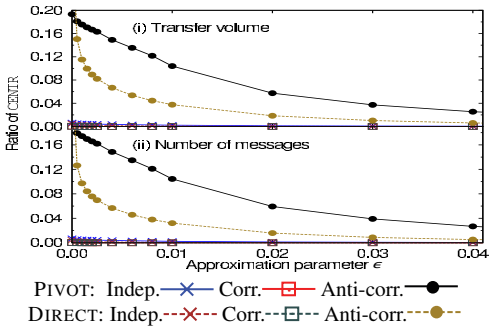

Fig. 9 Effect of approximation to cost of PIVOT and DIRECT

intervals. Y axis is interrupted at $y = 0.0065$ for illustration purposes. Clearly, both PIVOT and DIRECT enable substantial savings for all data sets. 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 since most objects end up close to the skyline border, leading to frequent skyline updates. Nevertheless, even for this data set, PIVOT and DIRECT still enable around 80% and 65% respectively network reduction compared to CENTR.

Also notice that DIRECT is more efficient than PIVOT for the streams with the correlated and independent dimensions, whereas PIVOT substantially outperforms DIRECT for the more challenging stream with anti-correlated dimensions. The reason for this discrepancy is the adaptivity extension, which is supported only by PIVOT. For the anti-correlated data set, the adaptivity extension switches around 7% of all objects to streaming monitoring (cf. Table 2). These are objects that end up to be very close to each other due to the anti-correlated dimensions. The effect of adaptive monitoring is not evident in the other two data sets, which induce far less threshold crossings, and are almost solely monitored using geometric monitoring. Since threshold queries of DIRECT are more compact than the ones of PIVOT, DIRECT turns out to be more efficient in the experiments with these two data sets.

Maximum Change. Streams up to now were generated assuming a maximum relative change maxRC $= 0.02$ per update. 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. 11(a) plots the network cost of PIVOT and DIRECT for data sets generated with independent dimensions. As expected, increasing maxRC results to an increase of the network cost of both algorithms. Nevertheless, even for maxRC $= 0.16$, both algorithms enable network savings of around 88% compared to CENTR. For small maxRC values, network savings of both algorithms are substantially higher, approximating 100%.

8.3 Approximate monitoring

All previous experiments considered exact monitoring, i.e., $\epsilon = 0$. Figure 9 plots the network cost in relation to ϵ for data sets with different correlations between dimensions. We see that an increase in the acceptable error can drastically reduce the

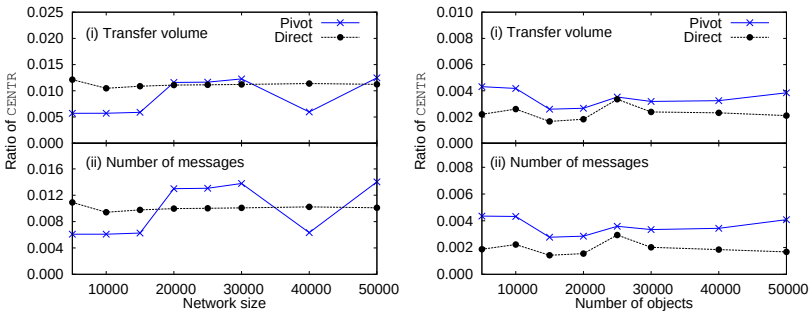

Fig. 10 Effect of (a) network size, (b) number of objects.

network cost. This is particularly visible in the experiments with anti-correlated dimensions where more threshold crossings are expected. We also observe that DIRECT utilizes error tolerance better, since it does not need to pre-allocate the error slack to objects, i.e., DIRECT allocates a total of ϵ tolerance per pair, whereas PIVOT pre-allocates the tolerance to $\epsilon/2$ per object. Notice that, as expected, the observed error in all experiments was always less than the guaranteed maximum error ϵ.

8.4 Scalability

To evaluate the scalability of the proposed algorithms, both algorithms were repeated on larger networks and with more objects. For the first series of experiments, we examined the cost of monitoring a fixed set of 1000 objects on networks of different sizes (all objects were monitored by all nodes). Fig. 10(a) plots the running cost of PIVOT and DIRECT as a ratio of CENTR for networks of up to 50.000 nodes. We see that the cost of both algorithms stays below 1.5% and presents no systematic increase with the network size.

For the second set of experiments, we examined the cost of monitoring object sets of different sizes over a fixed network of 1000 nodes. We again observe (Fig. 10(b)) that there is no systematic increase of network cost ratio with the number of objects. As such, our algorithm scales well with both number of objects and number of nodes.

8.5 Number and type of functions

The final set of experiments focused on investigating the influence of the number and type of functions to the performance of PIVOT and DIRECT.

Fig. 11(b) presents the performance of PIVOT and DIRECT when monitoring up to 5 linear functions. We observe that an increase of the number of functions leads to higher network cost for both algorithms. The main reason for this is that by adding functions, we increase the frequency of synchronizations (recall that a threshold violation on a single dimension is sufficient to invoke a synchronization). For up to four functions, PIVOT is substantially more efficient than CENTR, reducing the transfer volume by 70%, and the messages by more than 80%. For more than four functions, both PIVOT and DIRECT become less efficient than CENTR. We should note that skylines of higher dimensions are rarely considered in practical, real-world applications.

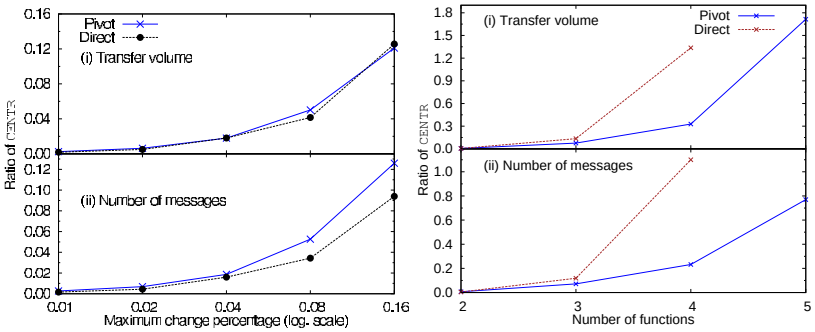

Fig. 11 Effect of (a) maximum relative change, (b) number of functions.

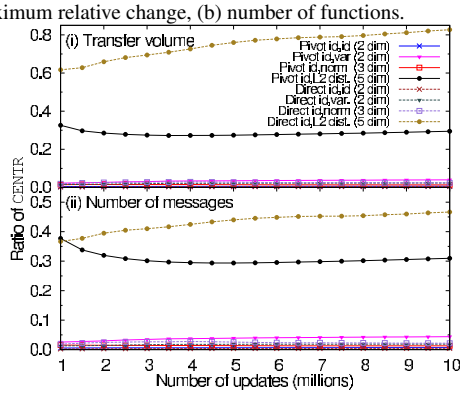

Fig. 12 Effect of types of functions.

The problem, of course, is that, in high dimensions, a large fraction of the objects end up in the skyline (due to the "dimensionality curse"), rendering it practically useless.

We also conducted experiments with more complex functions, namely the Euclidean norm on two dimensions ($Norm(\mathbf{v}(o,t)) = \sqrt{\sum_{i=1}^{2} \mathbf{v}(o,t)[i]^2}$), the L2 distance on four dimensions ($L2(\mathbf{v}(o,t)) = \sqrt{\sum_{i=1}^{2} \mathbf{v}(o,t)[i]^2 - \mathbf{v}(o,t)[i+2]^2}$), and the variance of one dimension on all sites ($Var(\mathbf{v}(o,t)[i]) = \sum_{p \in \mathcal{P}(o)} (\mathbf{v}(o,t,p)[i])^2 - \mathbf{v}(o,t)[i]^2$). 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. $\boldsymbol{f}[0]$ and $\boldsymbol{f}[1]$ were set to use different dimensions, in order to avoid introducing correlations.

Fig. 12 plots the network cost incurred by the two algorithms. For comparison, the figure also includes the cost for the case where both functions are set to the identity function. We see that the proposed algorithms substantially outperform CENTR, also on skylines defined through non-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 10 dimensions for L2 (cf. Section 4.1). This substantially increases the frequency of threshold crossings, and, consequently, 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.

8.6 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 operating on high 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. For the same reason, DIRECT also utilizes error tolerance better than PIVOT.

9 Conclusions

In this article we formally introduced the problem of continuous fragmented skyline queries. To address the problem, we proposed two distributed algorithms that rely on geometric monitoring to reduce the number of updates that need to be streamed to a central node, thereby reducing the total network cost for maintaining the skyline. We discussed several network optimizations for the two algorithms: (a) an approximation extension for error-tolerant applications, (b) grouping extensions which allow handling groups of objects more efficiently, and, (c) an adaptivity module which enables detecting highly volatile data and handling them more efficiently. The proposed algorithms were thoroughly evaluated with experiments on massive real-world and synthetic data sets. The experimental results demonstrated the scalability of the algorithm, as well as its significantly improved network efficiency compared to the only available baseline algorithm.

References

1. Babcock, B., Olston, C.: Distributed top-k monitoring. In: SIGMOD, pp. 28–39 (2003)
2. Balke, W.T., Gntzer, U., Zheng, J.X.: Efficient distributed skylining for web information systems. In: EDBT (2004)
3. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE (2001)
4. Cheema, M.A., Lin, X., Zhang, W., Zhang, Y.: A safe zone based approach for monitoring moving skyline queries. In: EDBT (2013)

5. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluation of probabilistic queries over imprecise data in constantly-evolving environments. Inf. Syst. **32**(1), 104–130 (2007)
6. Cormode, G., Garofalakis, M.: Approximate continuous querying over distributed streams. TODS **33**(2) (2008)
7. Cormode, G., Garofalakis, M., Muthukrishnan, S., Rastogi, R.: Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In: SIGMOD (2005)
8. Cranor, C., Johnson, T., Spatscheck, O., Shkapenyuk, V.: Gigascope: A stream database for network applications. In: SIGMOD (2003)
9. Cui, B., Lu, H., Xu, Q., Chen, L., Dai, Y., Zhou, Y.: Parallel distributed processing of constrained skyline queries by filtering. In: ICDE (2008)
10. Das, A., Ganguly, S., Garofalakis, M., Rastogi, R.: Distributed set-expression cardinality estimation. In: VLDB, pp. 312–323 (2004)
11. Graham, R., Knuth, D., Patashnik, O.: Concrete Mathematics: A Foundation for Computer Science. Addison-Wesley (1989)
12. HadjAli, A., Pivert, O., Prade, H.: On different types of fuzzy skylines. In: ISMIS 2011, pp. 581–591 (2011)
13. Hose, K., Vlachou, A.: A survey of skyline processing in highly distributed environments. VLDB J. (2011)
14. Huang, Z., Lu, H., Ooi, B.C., Tung, A.K.H.: Continuous skyline queries for moving objects. TKDE **18**(12) (2006)
15. Keren, D., Sharfman, I., Schuster, A., Livne, A.: Shape sensitive geometric monitoring. TKDE **24**(8) (2012)
16. Koltun, V., Papadimitriou, C.: Approximately dominating representatives. Theoretical Computer Science **371**(3), 148–154 (2007)
17. Lazerson, A., Sharfman, I., Keren, D., Schuster, A., Garofalakis, M.N., Samoladas, V.: Monitoring distributed streams using convex decompositions. PVLDB **8**(5), 545–556 (2015)
18. Lee, J., Hwang, S.: Scalable skyline computation using a balanced pivot selection technique. Inf. Syst. **39**, 1–21 (2014)
19. Madden, S., Franklin, M., Hellerstein, J., Hong, W.: The design of an acquisitional query processor for sensor networks. In: SIGMOD (2003)
20. Olston, C., Jiang, J., Widom, J.: Adaptive filters for continuous queries over distributed data streams. In: SIGMOD (2003)
21. Papadias, D., Fu, G., Chase, M., Seeger, B.: Progressive skyline computation in database systems. TODS **30**(1) (2005)
22. Papapetrou, O., Garofalakis, M.N.: Continuous fragmented skylines over distributed streams. In: ICDE (2014)
23. Sharfman, I., Schuster, A., Keren, D.: A geometric approach to monitoring threshold functions over distributed data streams. In: SIGMOD (2006)
24. Tao, Y., Papadias, D.: Maintaining sliding window skylines on data streams. TKDE **18**(2), 377–391 (2006)
25. Tao, Y., Xiao, X., Pei, J.: SUBSKY: Efficient computation of skylines in subspaces. In: ICDE (2006)
26. Trimponias, G., Bartolini, I., Papadias, D., Yang, Y.: Skyline processing on distributed vertical decompositions. TKDE **25**(4) (2013). DOI http://doi.ieeecomputersociety.org/10.1109/TKDE.2011.266
27. Vlachou, A., Doulkeridis, C., Kotidis, Y., Vazirgiannis, M.: Efficient routing of subspace skyline queries over highly distributed data. TKDE **22**(12) (2010)
28. Wu, P., Agrawal, D., Egecioglu, Ö., El Abbadi, A.: DeltaSky: Optimal maintenance of skyline deletions without exclusive dominance region generation. In: ICDE (2007)
29. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J.X., Zhang, Q.: Efficient computation of the skyline cube. In: VLDB (2005)
30. Zhang, S., Mamoulis, N., Cheung, D.W.: Scalable skyline computation using object-based space partitioning. In: SIGMOD (2009)
31. Zhang, Z., Cheng, R., Papadias, D., Tung, A.: Minimizing the communication cost for continuous skyline maintenance. In: SIGMOD (2009)