Approximate Geometric Query Tracking over Distributed Streams

Minos Garofalakis
School of Electronic and Computer Engineering
Technical University of Crete
minos@softnet.tuc.gr

Abstract

Effective Big Data analytics pose several difficult challenges for modern data management architectures. One key such challenge arises from the naturally streaming nature of big data, which mandates efficient algorithms for querying and analyzing massive, continuous data streams (that is, data that is seen only once and in a fixed order) with limited memory and CPU-time resources. Such streams arise naturally in emerging large-scale event monitoring applications; for instance, network-operations monitoring in large ISPs, where usage information from numerous sites needs to be continuously collected and analyzed for interesting trends. In addition to memory- and time-efficiency concerns, the inherently distributed nature of such applications also raises important communication-efficiency issues, making it critical to carefully optimize the use of the underlying network infrastructure. In this paper, we provide a brief introduction to the distributed data streaming model and the Geometric Method (GM), a generic technique for effectively tracking complex queries over massive distributed streams. We also discuss several recently-proposed extensions to the basic GM framework, such as the combination with stream-sketching tools and local prediction models, as well as more recent developments leading to a more general theory of Safe Zones and interesting connections to convex Euclidean geometry. Finally, we outline various challenging directions for future research in this area.

1 Introduction

Traditional data-management systems are typically built on a *pull-based paradigm*, where users issue one-shot queries to static data sets residing on disk, and the system processes these queries and returns their results. For several emerging application domains, however, data arrives and needs to be processed on a continuous (24×7) basis, without the benefit of several passes over a static, persistent data image. These *continuous data streams* arise naturally in new large-scale event monitoring applications, that require the ability to efficiently process continuous, high-volume streams of data in real time. Such monitoring systems are routinely employed, for instance, in the network installations of large Telecom and Internet service providers where detailed usage information (Call-Detail-Records (CDRs), SNMP/RMON packet-flow data, etc.) from different parts of the underlying network needs to be continuously collected and analyzed for interesting trends. Other examples

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

include real-time analysis tools for financial data streams, and event and operations monitoring applications for enterprise clouds and data centers. As both the scale of today's networked systems, and the volumes and rates of the associated data streams continue to increase with no bound in sight, algorithms and tools for effectively analyzing them are becoming an important research mandate.

Large-scale stream processing applications rely on *continuous*, event-driven monitoring, that is, real-time tracking of measurements and events, rather than one-shot answers to sporadic queries. Furthermore, the vast majority of these applications are inherently *distributed*, with several remote monitor sites observing their local, high-speed data streams and exchanging information through a communication network. This distribution of the data naturally implies critical communication constraints that typically prohibit centralizing all the streaming data, due to either the huge volume of the data (e.g., in IP-network monitoring, where the massive amounts of collected utilization and traffic information can overwhelm the production IP network [13]), or power and bandwidth restrictions (e.g., in wireless sensornets, where communication is the key determinant of sensor battery life [28]). Finally, an important requirement of large-scale event monitoring is the effective support for tracking complex, *holistic queries* that provide a global view of the data by combining and correlating information across the collection of remote monitor sites. For instance, tracking aggregates over the result of a distributed join (the "workhorse" operator for combining tables in relational databases) can provide unique, real-time insights into the workings of a large-scale distributed system, including system-wide correlations and potential anomalies [7]. Monitoring the precise value of such holistic queries without continuously centralizing all the data seems hopeless; luckily, when tracking statistical behavior and patters in large scale systems, *approximate answers* (with reasonable approximation error guarantees) are often sufficient. This often allows algorithms to effectively tradeoff efficiency with approximation quality (e.g., using sketch-based stream approximations [7]).

Given the prohibitive cost of data centralization, it is clear that realizing sophisticated, large-scale distributed data-stream analysis tools must rely on novel algorithmic paradigms for processing local streams of data *in situ* (i.e., locally at the sites where the data is observed). This, of course, implies the need for intelligently decomposing a (possibly complex) global data-analysis and monitoring query into a collection of "safe" local queries that can be tracked independently at each site (without communication), while guaranteeing correctness for the global monitoring operation. This decomposition process can enable truly distributed, event-driven processing of real-time streaming data, using a *push-based paradigm*, where sites monitor their local queries and communicate only when some local query constraints are violated [7, 34]. Nevertheless, effectively decomposing a complex, holistic query over the global collections of streams into such local constraints is far from straightforward, especially in the case of *non-linear* queries (e.g., norms or joins) [34].

The bulk of early work on data-stream processing has focused on developing space-efficient, one-pass algorithms for performing a wide range of *centralized computations* on massive data streams; examples include computing quantiles [22], estimating distinct values [20], and set-expression cardinalities [16], counting frequent elements (i.e., "heavy hitters") [5, 11, 29], approximating large Haar-wavelet coefficients [10], and estimating join sizes and stream norms [1, 2, 15]. Monitoring *distributed* data streams has attracted substantial research interest in recent years [6, 31], with early work focusing on the monitoring of *single values*, and building appropriate models and filters to avoid propagating updates if these are insignificant compared to the value of simple *linear* aggregates (e.g., to the `SUM` of the distributed values). For instance, [32] proposes a scheme based on "adaptive filters" — that is, bounds around the value of distributed variables, which shrink or grow in response to relative stability or variability, while ensuring that the total uncertainty in the bounds is at most a user-specified bound. Still, in the case of linear aggregate functions, deriving local filter bounds based on a global monitoring condition is rather straightforward, with the key issue being how to intelligently distribute the available aggregate "slack" across all sites [3, 9, 24].

In this paper, we focus on recently-developed algorithmic tools for effectively tracking a broad class of complex queries over massive, distributed data streams. We start by describing the key elements of a generic *distributed stream-processing model* and define a broad class of distributed query-tracking problems addressed by our techniques. We then give an overview of the *Geometric Method (GM)* [34, 25] for distributed threshold

monitoring that lies at the core of our distributed query-tracking methodology, and briefly discuss recent extensions to the basic GM framework that incorporate stream sketches [17] and local prediction models [18, 19]. We also summarize recent developments leading to a more general theory of *Safe Zones* for geometric monitoring and interesting connections to convex Euclidean geometry [27]. Finally, we conclude with a brief discussion of new research directions in this space.

2 Distributed Data Streaming and the Geometric Method

Data-Stream Processing. Recent years have witnessed an increasing interest in designing data-processing algorithms that work over continuous data streams, i.e., algorithms that provide results to user queries while looking at the relevant data items *only once and in a fixed order* (determined by the stream-arrival pattern). Data-stream processing turns the paradigm of conventional database systems on its head: Databases typically have to deal with a stream of queries over a static, bounded data set; instead, a stream processing engine has to effectively process a static set of queries over continuous streams of data. Such stream queries are typically *continuous*, implying the need for continuous, real-time monitoring of the query answer over the changing stream.

Formally, a data stream can be modeled as a *massive, dynamic, one-dimensional vector* $\boldsymbol{v}[1 \ldots N]$ that, at any point in time, captures the current state of the stream. Note that this is a very generic, powerful model — for instance, in the case of streams of relational tuples (rendering a dynamic relational table), this vector $\boldsymbol{v}$ is essentially the (dynamic) frequency distribution vector of the underlying relational table whose values capture the counts of different tuples (i.e., attribute value combinations) in the relation. (Multi-attribute relational tables can naturally be handled in this abstract model by simply "unfolding" the corresponding multi-dimensional frequency distribution on one vector dimension using standard techniques, e.g., row- or column-major). As an example, in the case of IP routers monitoring the number of `TCP` connections and `UDP` packets exchanged between source and destination IP addresses, the stream vector $\boldsymbol{v}$ has 2×2^{64} entries capturing the up-to-date frequencies for specific (source, destination) pairs observed in `TCP` connections and `UDP` packets routed through router j. (For instance, the first (last) 2^{64} entries of $\boldsymbol{v}$ could be used for TCP-connection (respectively, UDP-packet) frequencies.) The size N of the stream vector $\boldsymbol{v}$ vector is defined as the product of the attribute domain size(s) which can easily grow very large. [1] The dynamic vector $\boldsymbol{v}$ is rendered through a continuous stream of updates, where each update effectively modifies values in $\boldsymbol{v}$ — the nature of these update operations gives rise to different data streaming models, such as *time-series*, *cash-register*, and *turnstile* streams [30].

Data-stream processing algorithms aim to compute functions (or, queries) on the stream vector $\boldsymbol{v}$ at different points during the lifetime of the stream (continuous or ad-hoc). Since N can be very large, the typical requirement here is that these algorithms work in *small space* (i.e., the state maintained by the algorithm) and *small time* (i.e., the processing time per update), where "small" is understood to mean a quantity significantly smaller than $\Theta(N)$ (typically, poly-logarithmic in N). Several such stream-processing algorithms are known for various data-analysis queries [1, 2, 5, 10, 11, 15, 16, 20, 22, 29].

Distributed Data Streaming. The naturally distributed nature of large-scale event-monitoring applications (such as the ones mentioned earlier) implies one additional level of complexity, in the sense that there is no centralized observation point for the dynamic stream vector $\boldsymbol{v}$; instead, $\boldsymbol{v}$ is distributed across several sites. More specifically, we consider a distributed computing environment, comprising a collection of k *remote sites* and a designated *coordinator site*. Streams of data updates arrive continuously at remote sites, while the coordinator site is responsible for generating approximate answers to (possibly, continuous) user queries posed over the collection of remotely-observed streams (across all sites). Following earlier work in the area [3, 7, 9, 14, 32], our distributed stream-processing model does not explicitly allow direct communication between remote sites;

[1] Note that streaming algorithms typically do not require a priori knowledge of N.

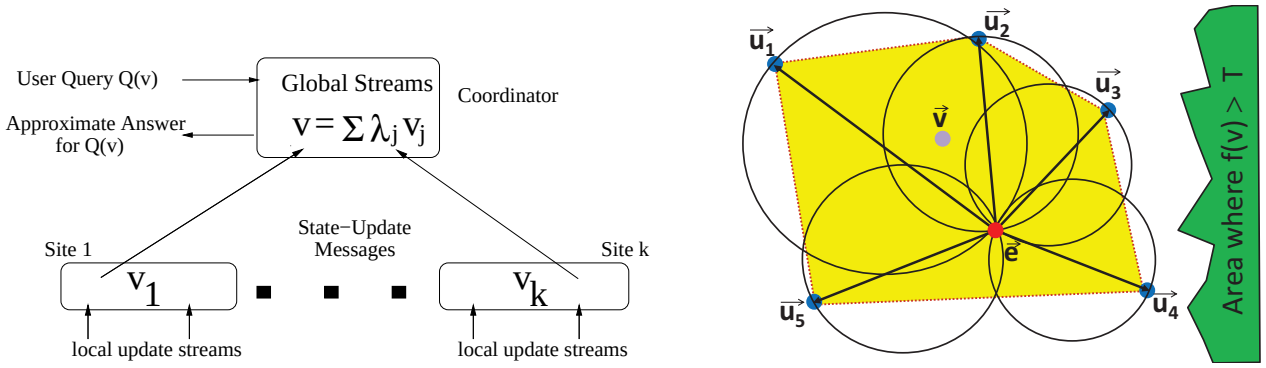

Figure 1: (a) Distributed stream processing architecture. (b) Geometric Method: Estimate vector $\vec{e}$, drift vectors $\boldsymbol{u}_j$, convex hull enclosing current $\boldsymbol{v}$ (dotted outline), and bounding balls $B(\boldsymbol{e} + \frac{1}{2}\Delta\boldsymbol{v}_j, \frac{1}{2}\|\Delta\boldsymbol{v}_j\|)$.

instead, as illustrated in Figure 1(a), a remote site exchanges messages only with the coordinator, providing it with state information on its (locally-observed) streams. [2] Note that such a hierarchical processing model is, in fact, representative of a large class of applications, including network monitoring where a central Network Operations Center (NOC) is responsible for processing network traffic statistics (e.g., link bandwidth utilization, IP source-destination byte counts) collected at switches, routers, and/or Element Management Systems (EMSs) distributed across the network.

Each remote site $j \in \{1, \ldots, k\}$ observes (possibly, several) local update streams that incrementally render a *local stream vector* $\boldsymbol{v}_j$ capturing the current local state of the observed stream(s) at site j. All local stream vectors $\boldsymbol{v}_j$ in our distributed streaming architecture change dynamically over time — when necessary, we make this dependence explicit, using $\boldsymbol{v}_j(t)$ to denote the state of the vector at time t (assuming a consistent notion of "global time" in our distributed system). The unqualified notation $\boldsymbol{v}_j$ typically refers to the *current* state of the local stream vector.

We define the *global stream vector* $\boldsymbol{v}$ of our distributed stream(s) as any *weighted average* (i.e., convex combination) of the local stream vectors $\{\boldsymbol{v}_j\}$; that is, $\boldsymbol{v} = \sum_{j=1}^{k} \lambda_j \boldsymbol{v}_j$, where $\sum_j \lambda_j = 1$ and $\lambda_j \geq 0$ for all j. (Again, to simplify notation, we typically omit the explicit dependence on time when referring to the current global vector.) Our focus is on the problem of effectively answering user queries (or, functions) over the global stream vector at the coordinator site. Rather than one-time query/function evaluation, we assume a continuous-querying environment which implies that the coordinator needs to *continuously maintain* (or, *track*) the answers to queries as the local update streams $\boldsymbol{v}_j$ evolve at individual remote sites. There are two defining characteristics of our problem setup that raise difficult algorithmic challenges for our query tracking problems:

• *The distributed nature and large volumes of local streaming data* raise important communication and space/-time efficiency concerns. Naïve schemes that accurately track query answers by forcing remote sites to ship every remote stream update to the coordinator are clearly impractical, since they can impose an inordinate burden on the underlying communication infrastructure (especially, for high-rate data streams and large numbers of remote sites). Furthermore, the voluminous nature of the local data streams implies that effective streaming tools are needed at the remote sites in order to manage the local stream vectors in sublinear space/time. Thus, a practical approach is to adopt the paradigm of continuous tracking of *approximate* query answers at the coordinator site with strong guarantees on the quality of the approximation. This allows schemes that can effectively trade-off space/time/communication efficiency and query-approximation accuracy in a precise, quantitative manner.

• *General, non-linear queries/functions* imply fundamental and difficult challenges for distributed monitoring.

[2] Of course, sites can always communicate with each other through the coordinator — this would only increase communication load by a factor of 2.

For the case of linear functions, a number of approaches have been proposed that rely on the key idea of allocating appropriate *"slacks"* to the remote sites based on their locally-observed function values (e.g., [3, 32, 24]). Unfortunately, it is not difficult to find examples of simple *non-linear* functions on one-dimensional data, where it is basically impossible to make any assumptions about the value of the global function based on the values observed locally at the sites [34]. This renders conventional slack-allocation schemes inapplicable in this more general setting.

The Geometric Method (GM). Sharfman et al. [34] consider the fundamental problem of *distributed threshold monitoring*; that is, determine whether $f(\boldsymbol{v}) < \tau$ or $f(\boldsymbol{v}) > \tau$, for a given (general) function $f()$ over the global stream vector and a fixed threshold τ. Their key idea is that, since it is generally impossible to connect the locally-observed values of $f()$ to the global value $f(\boldsymbol{v})$, one can employ geometric arguments to monitor the *domain* (rather than the range) of the monitored function $f()$. More specifically, assume that at any point in time, each site j has informed the coordinator of some prior state of its local vector $\boldsymbol{v}_j^p$; thus, the coordinator has an estimated global vector $\boldsymbol{e} = \boldsymbol{v}^p = \sum_{j=1}^{k} \lambda_j \boldsymbol{v}_j^p$. Clearly, the updates arriving at sites can cause the local vectors $\boldsymbol{v}_j$ to drift too far from their previously reported values $\boldsymbol{v}_j^p$, possibly leading to a violation of the τ threshold. Let $\Delta\boldsymbol{v}_j = \boldsymbol{v}_j - \boldsymbol{v}_j^p$ denote the local *delta vector* (due to updates) at site j, and let $\boldsymbol{u}_j = \boldsymbol{e} + \Delta\boldsymbol{v}_j$ be the *drift vector* from the previously reported estimate at site j. We can then express the current global stream vector $\boldsymbol{v}$ in terms of the drift vectors:

$$\boldsymbol{v} \;=\; \sum_{j=1}^{k} \lambda_j (\boldsymbol{v}_j^p + \Delta\boldsymbol{v}_j) \;=\; \boldsymbol{e} + \sum_{j=1}^{k} \lambda_j \Delta\boldsymbol{v}_j \;=\; \sum_{j=1}^{k} \lambda_j (\boldsymbol{e} + \Delta\boldsymbol{v}_j).$$

That is, the current global vector is a convex combination of drift vectors and, thus, guaranteed to lie somewhere within the convex hull of the delta vectors around $\boldsymbol{e}$. Figure 1(b) depicts an example in $d = 2$ dimensions. The current value of the global stream vector lies somewhere within the shaded convex-hull region; thus, as long as the convex hull does not overlap the inadmissible region (i.e., the region $\{\boldsymbol{v} \in \mathbb{R}^2 : f(\boldsymbol{v}) > \tau\}$ in Figure 1(b)), we can guarantee that the threshold has not been violated (i.e., $f(\boldsymbol{v}) \leq \tau$)).

The problem, of course, is that the $\Delta\boldsymbol{v}_j$'s are spread across the sites and, thus, the above condition cannot be checked locally. To transform the global condition into a local constraint, we place a d-dimensional *bounding ball* $B(\boldsymbol{c}, r)$ around each local delta vector, of radius $r = \frac{1}{2}\|\Delta\boldsymbol{v}_j\|$ and centered at $\boldsymbol{c} = \boldsymbol{e} + \frac{1}{2}\Delta\boldsymbol{v}_j$ (see Figure 1(b)). It can be shown that the union of all these balls completely covers the convex hull of the drift vectors [34]. This observation effectively reduces the problem of monitoring the global stream vector to the local problem of each remote site monitoring the ball around its local delta vector.

More specifically, given the monitored function $f()$ and threshold τ, we can partition the d-dimensional space into two sets $A = \{\boldsymbol{v} : f(\boldsymbol{v}) \leq \tau\}$ and $\overline{A} = \{\boldsymbol{v} : f(\boldsymbol{v}) > \tau\}$. (Note that these sets can be arbitrarily complex, e.g., they may comprise multiple disjoint regions of $\mathbb{R}^d$.) The basic protocol is now quite simple: Each site monitors its delta vector $\Delta\boldsymbol{v}_j$ and, with each update, checks whether its bounding ball $B(\boldsymbol{e} + \frac{1}{2}\Delta\boldsymbol{v}_j, \frac{1}{2}\|\Delta\boldsymbol{v}_j\|)$ is *monochromatic*, i.e., all points in the ball lie within the same region (A or $\overline{A}$). If this is not the case, we have a *local threshold violation*, and the site communicates its local $\Delta\boldsymbol{v}_j$ to the coordinator. The coordinator then initiates a *synchronization process* that typically tries to resolve the local violation by communicating with only a subset of the sites in order to "balance out" the violating $\Delta\boldsymbol{v}_j$ and ensure the monochromicity of all local bounding balls [34]. In the worst case, the delta vectors from all k sites are collected, leading to an accurate estimate of the current global stream vector, which is by definition monochromatic (since all bounding balls have 0 radius).

The power of the GM stems from the fact that it is essentially agnostic of the specific (global) function $f(\boldsymbol{v})$ being monitored.[3] Note that the function itself is only used at a remote site when checking the monochromicity

[3]The assumption that GM only monitors functions of the (weighted) average of local stream vectors is not really restrictive: Numerous complex functions can actually be expressed as functions of the average using simple tricks, such as adding additional dimensions to the stream vectors, e.g., [4].

of its local ball, which essentially boils down to solving a minimization/maximization problem for $f()$ within the area of that ball. This may, of course, be complex but it also enables the GM to effectively trade local computation for communication.

From Threshold Crossing to Approximate Query Tracking. Consider the task of monitoring (at the coordinator) the value of a function $f()$ over the global stream vector $\boldsymbol{v}$ to within θ relative error. (Our discussion here focuses on relative error – the case of monitoring to within bounded *absolute* error can be handled in a similar manner.) Since all the coordinator has is the estimated value of the global stream vector $\boldsymbol{e} = \boldsymbol{v}^p$ based on the most recent site updates $\boldsymbol{v}_j^p$, our monitoring protocol would have to guarantee that the estimated function value carries at most θ relative error compared to the up-to-date value $f(\boldsymbol{v}) = f(\boldsymbol{v}(t))$, that is $f(\boldsymbol{v}^p) \in (1 \pm \theta) f(\boldsymbol{v})$ [4], which is obviously equivalent to monitoring two threshold queries on $f(\boldsymbol{v})$:

$$f(\boldsymbol{v}) \geq \frac{f(\boldsymbol{v}^p)}{1+\theta} \quad \text{and} \quad f(\boldsymbol{v}) \leq \frac{f(\boldsymbol{v}^p)}{1-\theta}.$$

These are exactly the threshold conditions that our approximate function tracking protocols will need to monitor. Note that $f(\boldsymbol{v}^p)$ in the above expression is a constant (based on the latest communication of the coordinator with the remote sites). Similar threshold conditions can also be derived when the local/global values of $f()$ are only known approximately (e.g., using sketches [1, 2] or other streaming approximations) — the threshold conditions just need to account for the added approximation error [17].

3 Enhancing GM: Sketches and Prediction Models

In this section, we give an overview of more recent work on extending the GM with two key stream-processing tools, namely sketches and prediction models [17, 18, 19].

GM and AMS Sketches. Techniques based on small-space pseudo-random *sketch* summaries of the data have proved to be very effective tools for dealing with massive, rapid-rate data streams in centralized settings [8]. The key idea in such sketching techniques is to represent a streaming frequency vector $\boldsymbol{v}$ using a much smaller (typically, randomized) *sketch* vector (denoted by $\texttt{sk}(\boldsymbol{v})$) that (1) can be easily maintained as the updates incrementally rendering $\boldsymbol{v}$ are streaming by, and (2) provide probabilistic guarantees for the quality of the data approximation. The widely used AMS sketch (proposed by Alon, Matias, and Szegedy in their seminal paper [2]) defines the entries of the sketch vector as pseudo-random linear projections of $\boldsymbol{v}$ that can easily maintained over the stream of updates. AMS sketch estimators can effectively approximate *inner-product queries* $\boldsymbol{v} \cdot \boldsymbol{u} = \sum_i \boldsymbol{v}[i] \cdot \boldsymbol{u}[i]$ over streaming data vectors and tensors. Such inner products naturally map to several interesting query classes, including join and multi-join aggregates [15], range and quantile queries [21], heavy hitters and top-k queries [5], and approximate histogram and wavelet representations [10]. The AMS estimator function $f_{\text{AMS}}()$ computed over the sketch vectors of $\boldsymbol{v}$ and $\boldsymbol{u}$ is complex, and involves both averaging and median-selection over the components of the sketch-vector inner product [1, 2]. Formally, viewing each sketch vector as a two-dimensional $n \times m$ array (where $n = O(\frac{1}{\epsilon^2})$, $m = O(\log(1/\delta))$ and ϵ, $1-\delta$ denote desired bounds on error and probabilistic confidence (respectively)), the AMS estimator function is defined as:

$$f_{\text{AMS}}(\texttt{sk}(\boldsymbol{v}), \texttt{sk}(\boldsymbol{u})) \;=\; \underset{i=1,\ldots,m}{\texttt{median}} \left\{ \frac{1}{n} \sum_{l=1}^{n} \texttt{sk}(\boldsymbol{v})[l,i] \cdot \texttt{sk}(\boldsymbol{u})[l,i] \right\}, \tag{14}$$

and guarantees that, with probability $\geq 1-\delta$, $f_{\text{AMS}}(\texttt{sk}(\boldsymbol{v}), \texttt{sk}(\boldsymbol{u})) \in (\boldsymbol{v} \cdot \boldsymbol{u} \pm \epsilon \|\boldsymbol{v}\| \|\boldsymbol{u}\|)$ [1, 2].

Moving to the distributed streams setting, note that our discussion of the GM thus far has assumed that all remote sites maintain the *full stream vector* (i.e., employ $\Theta(N)$ space), which is often unrealistic for real-life data streams. In our recent work [17], we have proposed novel approximate query tracking protocols that

[4] Throughout, the notation $x \in (y \pm z)$ is equivalent to $|x-y| \leq |z|$.

exploit the combination of the GM and AMS sketch estimators. The AMS sketching idea offers an effective streaming dimensionality-reduction tool that significantly expands the scope of the original GM, allowing it to handle massive, high-dimensional distributed data streams in an efficient manner with approximation-quality guarantees. Furthermore, the linearity of AMS sketches implies that they can be trivially merged (by simple component-wise addition), making them particularly suitable to our distributed streams settings [7]. A key technical observation is that, by exploiting properties of the AMS estimator function, geometric monitoring can now take place in a *much lower-dimensional space*, allowing for communication-efficient monitoring. Another technical challenge that arises is how to effectively test the monochromicity of bounding balls in this lower-dimensional space with respect to threshold conditions involving the highly non-linear `median` operator in the AMS estimator $f_{\text{AMS}}()$ (Equation (14)). We have proposed a number of novel algorithmic techniques to address these technical challenges, starting from the easier cases of L_2-norm (i.e., self-join) and range queries, and then extending them to the case of general inner-product (i.e., binary-join) queries. Our experimental study with real-life data sets demonstrates the practical benefits of our approach, showing consistent gains of up to 35% in terms of total communication cost compared to state-of-the-art methods; furthermore, our techniques demonstrate even more impressive benefits (of over 100%) when focusing on the communication costs of data (i.e., sketch) shipping in the system.

GM and Prediction Models. In other recent work [18, 19], we have proposed a novel combination of the geometric method with *local prediction models* for describing the temporal evolution of local data streams. (The adoption of prediction models has already been proven beneficial in terms of bandwidth preservation in distributed settings [7].) We demonstrate that prediction models can be incorporated in a very natural way in the geometric method for tracking general, non-linear functions; furthermore, we show that the initial geometric monitoring method of Sharfman et al. [25, 34] is only a special case of our, more general, prediction-based geometric monitoring framework. Interestingly, the mere utilization of local predictions is not enough to guarantee lower communication overheads even when predictors are quite capable of describing local stream distributions. We establish a theoretically solid monitoring framework that incorporates conditions that can lead to fewer contacts with the coordinator. We also develop a number of mechanisms, along with extensive probabilistic models and analysis, that relax the previously introduced framework, base their function on simpler criteria, and yield significant communication benefits in practical scenarios.

4 Towards Convex Safe Zones

In followup work to the GM, Keren et al. [25] propose a simple, generic geometric monitoring strategy that can be formally shown to encompass the original GM scheme as a special case. Briefly, assuming we are monitoring the threshold condition $f(\boldsymbol{v}) \leq \tau$, the idea is to define a certain *convex subset* C of the admissible region $A = \{\boldsymbol{v} : f(\boldsymbol{v}) \leq \tau\}$ (i.e., a *convex admissible subset*), which is then used to define *Safe Zones (SZs)* for the local drift vectors: *Site j simply monitors the condition* $\boldsymbol{u}_j = \boldsymbol{e} + \Delta\boldsymbol{v}_j \in C$. The correctness of this generic monitoring scheme follows directly from the convexity of C, and our earlier observation that the global stream vector $\boldsymbol{v}$ always lies in the convex hull of $\boldsymbol{u}_j$, $j = 1, \ldots, k$: If $\boldsymbol{u}_j \in C$ for all nodes j then, by convexity, this convex hull (and, therefore $\boldsymbol{v}$) lies completely within C and, therefore, the admissible region (since $C \subseteq A$). (Note that the convexity of C plays a crucial role in the above correctness argument.)

While the convexity of C is needed for the *correctness* of the monitoring scheme, it is clear that the size of C plays a critical role in its *efficiency*: Obviously, a larger C implies fewer local violations and, thus, smaller communication/synchronization overheads. This, in turn, implies a fairly obvious dominance relationship over geometric distributed monitoring schemes: Given two geometric algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ (for the same distributed monitoring problem) that use the convex admissible subsets C_1 and C_2 (respectively), algorithm $\mathcal{A}_1$ is *provably superior* to $\mathcal{A}_2$ if $C_2 \subset C_1$. Note that, in the simple case of *linear* functions $f()$, the admissible region A itself is convex, and therefore one can choose $C = A$; however, for more complicated, non-linear functions, A is

non-convex and quite complex. Thus, finding a "large" convex subset of A is a crucial component of effective geometric monitoring.

Interestingly, the bounding ball constraints of the GM can also be cast in terms of a convex admissible subset (denoted by C_{GM}) that can be mathematically shown to be equivalent to the intersection of the (possibly, infinitely many) half-spaces defined by points at the boundary of the admissible region A [25]. Furthermore, as demonstrated in our recent work [27], while the GM can achieve good results and is generic (i.e., can be applied to any monitoring function), its performance can be far from optimal since its underlying SZ C_{GM} is often far too restrictive. In several practical scenarios, C_{GM} can be drastically improved by intersecting *much fewer half-spaces* in order to obtain provably larger convex admissible subsets, giving significantly more efficient monitoring schemes. In a nutshell, our proposed *Convex Decomposition (CD)* method works by identifying convex subsets of the *inadmissible region*, and using them to define non-redundant collections of half-spaces that separate these subsets from the admissible region [27]. Our CD methodology can be applied to several important approximate query monitoring tasks (e.g., norms, range aggregates, and joins) giving provably larger SZs and substantially better performance than the original GM.

5 Conclusions and Future Directions

We have given a brief introduction to the distributed data streaming model and the Geometric Method (GM), a generic technique for effectively tracking complex queries over massive distributed streams. We have also discussed recently-proposed extensions to the basic GM framework, such as the combination with AMS stream sketches and local prediction models, as well as recent developments leading to a more general theory of *Safe Zones* for geometric monitoring and interesting connections to convex Euclidean geometry. The GM framework provides a very powerful tool for dealing with continuous query computations over distributed streaming data; see, for instance, [33] for a novel application of the GM to continuous monitoring of skyline queries over fragmented dynamic data.

Continuous distributed streaming is a vibrant, rapidly evolving field of research, and a community of researchers has started forming around theoretical, algorithmic, and systems issues in the area [31] Naturally, there are several promising directions for future research. First, the single-level hierarchy model (depicted in Figure 1(a)) is simplistic and also introduces a single point of failure (i.e., the coordinator). Extending the model to general hierarchies is probably not that difficult (even though effectively distributing the error bounds across the internal hierarchy nodes can be challenging [7]); however, extending the ideas to general, scalable distributed architectures (e.g., P2P networks) raises several theoretical and practical challenges. Second, while most of the proposed algorithmic tools have been prototyped and tested with real-life data streams, there is still a need for real system implementations that also address some of the key systems questions that arise (e.g., what functions and query language to support, how to interface to real users and applications, and so on). We have already started implementing some of the geometric monitoring ideas using Twitter's Storm/λ-architecture, and exploiting these ideas for large-scale, distributed Complex Event Processing (CEP) in the context of the FERARI project (`www.ferari-project.eu`). Finally, from a more foundational perspective, there is a need for developing new models and theories for studying the complexity of such *continuous distributed computations*. These could build on the models of *communication complexity* [26] that study the complexity of distributed *one-shot* computations, perhaps combined with relevant ideas from information theory (e.g., distributed source coding). Some initial results in this direction have recently appeared for the case of simple norms and linear aggregates, e.g., [12, 23].

Acknowledgements. This work was partially supported by the European Commission under ICT-FP7-FERARI (Flexible Event Processing for Big Data Architectures), `www.ferari-project.eu`.

References

[1] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. "Tracking Join and Self-Join Sizes in Limited Storage". In *Proc. of the 18th ACM Symposium on Principles of Database Systems*, May 1999.

[2] N. Alon, Y. Matias, and M. Szegedy. "The Space Complexity of Approximating the Frequency Moments". In *Proc. of the 28th Annual ACM Symposium on the Theory of Computing*, May 1996.

[3] B. Babcock and C. Olston. "Distributed Top-K Monitoring". In *Proc. of the 2003 ACM SIGMOD Intl. Conference on Management of Data*, June 2003.

[4] S. Burdakis and A. Deligiannakis. "Detecting Outliers in Sensor Networks Using the Geometric Approach". In *Proc. of the 28th Intl. Conference on Data Engineering*, Apr. 2012.

[5] M. Charikar, K. Chen, and M. Farach-Colton. "Finding Frequent Items in Data Streams". In *Proc. of the Intl. Colloquium on Automata, Languages, and Programming*, July 2002.

[6] G. Cormode and M. Garofalakis. "Streaming in a Connected World: Querying and Tracking Distributed Data Streams". Tutorial in *2007 ACM SIGMOD Intl. Conf. on Management of Data*, June 2007.

[7] G. Cormode and M. Garofalakis. "Approximate Continuous Querying over Distributed Streams". *ACM Transactions on Database Systems*, 33(2), June 2008.

[8] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. "Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches". *Foundations and Trends in Databases*, 4(1-3), 2012.

[9] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. "Holistic Aggregates in a Networked World: Distributed Tracking of Approximate Quantiles". In *Proc. of the 2005 ACM SIGMOD Intl. Conference on Management of Data*, June 2005.

[10] G. Cormode, M. Garofalakis, and D. Sacharidis. "Fast Approximate Wavelet Tracking on Streams". In *Proc. of the 10th Intl. Conference on Extending Database Technology (EDBT'2006)*, Mar. 2006.

[11] G. Cormode and S. Muthukrishnan. "What's Hot and What's Not: Tracking Most Frequent Items Dynamically". In *Proc. of the 22nd ACM Symposium on Principles of Database Systems*, June 2003.

[12] G. Cormode, S. Muthukrishnan, and K. Yi. "Algorithms for distributed functional monitoring". *ACM Transactions on Algorithms*, 7(2), 2011.

[13] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. "Gigascope: A Stream Database for Network Applications". In *Proc. of the 2003 ACM SIGMOD Intl. Conference on Management of Data*, June 2003.

[14] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. "Distributed Set-Expression Cardinality Estimation". In *Proc. of the 30th Intl. Conference on Very Large Data Bases*, Sept. 2004.

[15] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. "Processing Complex Aggregate Queries over Data Streams". In *Proc. of the 2002 ACM SIGMOD Intl. Conference on Management of Data*, June 2002.

[16] S. Ganguly, M. Garofalakis, and R. Rastogi. "Processing Set Expressions over Continuous Update Streams". In *Proc. of the 2003 ACM SIGMOD Intl. Conference on Management of Data*, June 2003.

[17] M. Garofalakis, D. Keren, and V. Samoladas. "Sketch-based Geometric Monitoring of Distributed Stream Queries". In *Proc. of the 39th Intl. Conference on Very Large Data Bases*, Aug. 2013.

[18] N. Giatrakos, A. Deligiannakis, M. Garofalakis, I. Sharfman, and A. Schuster. "Prediction-based Geometric Monitoring of Distributed Data Streams". In *Proc. of the 2012 ACM SIGMOD Intl. Conference on Management of Data*, May 2012.

[19] N. Giatrakos, A. Deligiannakis, M. Garofalakis, I. Sharfman, and A. Schuster. "Distributed Geometric Query Monitoring using Prediction Models". *ACM Transactions on Database Systems*, 39(2), May 2014.

[20] P. B. Gibbons. "Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports". In *Proc. of the 27th Intl. Conference on Very Large Data Bases*, Sept. 2001.

[21] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. "How to Summarize the Universe: Dynamic Maintenance of Quantiles". In *Proc. of the 28th Intl. Conference on Very Large Data Bases*, Aug. 2002.

[22] M. B. Greenwald and S. Khanna. "Space-Efficient Online Computation of Quantile Summaries". In *Proc. of the 2001 ACM SIGMOD Intl. Conference on Management of Data*, May 2001.

[23] Z. Huang, K. Yi, and Q. Zhang. "Randomized algorithms for tracking distributed count, frequencies, and ranks". In *Proc. of the 31st ACM Symposium on Principles of Database Systems*, May 2012.

[24] R. Keralapura, G. Cormode, and J. Ramamirtham. "Communication-efficient distributed monitoring of thresholded counts". In *Proc. of the 2006 ACM SIGMOD Intl. Conference on Management of Data*, June 2006.

[25] D. Keren, I. Sharfman, A. Schuster, and A. Livne. "Shape-Sensitive Geometric Monitoring". *IEEE Transactions on Knowledge and Data Engineering*, 24(8), Aug. 2012.

[26] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[27] A. Lazerson, I. Sharfman, D. Keren, A. Schuster, M. Garofalakis, and V. Samoladas. "Monitoring Distributed Streams using Convex Decompositions". In *Proc. of the 41st Intl. Conference on Very Large Data Bases*, Aug. 2015.

[28] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. "The Design of an Acquisitional Query Processor for Sensor Networks". In *Proc. of the 2003 ACM SIGMOD Intl. Conference on Management of Data*, June 2003.

[29] G. S. Manku and R. Motwani. "Approximate Frequency Counts over Data Streams". In *Proc. of the 28th Intl. Conference on Very Large Data Bases*, Aug. 2002.

[30] S. Muthukrishnan. "Data Streams: Algorithms and Applications". *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[31] NII Shonan Workshop on Large-Scale Distributed Computation, Shonan Village, Japan, January 2012. `http://www.nii.ac.jp/shonan/seminar011/`.

[32] C. Olston, J. Jiang, and J. Widom. "Adaptive Filters for Continuous Queries over Distributed Data Streams". In *Proc. of the 2003 ACM SIGMOD Intl. Conference on Management of Data*, June 2003.

[33] O. Papapetrou and M. Garofalakis. "Continuous Fragmented Skylines over Distributed Streams". In *Proc. of the 30th Intl. Conference on Data Engineering*, Apr. 2014.

[34] I. Sharfman, A. Schuster, and D. Keren. "A geometric approach to monitoring threshold functions over distributed data streams". In *Proc. of the 2006 ACM SIGMOD Intl. Conference on Management of Data*, June 2006.