One for All and All for One: Simultaneous Approximation of Multiple Functions over Distributed Streams

Arnon Lazerson
Moshe Gabel
lazerson@cs.technion.ac.il
mgabel@cs.technion.ac.il
Technion
Haifa 32000 Israel

Daniel Keren
dkeren@cs.haifa.ac.il
University of Haifa
Haifa 31905 Israel

Assaf Schuster
assaf@cs.technion.ac.il
Technion
Haifa 32000 Israel

ABSTRACT

Distributed monitoring methods address the difficult problem of continuously approximating functions over distributed streams, while minimizing the communication cost. However, existing methods are concerned with the approximation of a single function at a time. Employing these methods to track multiple functions will multiply the communication volume, thus eliminating their advantage in the first place.

We introduce a novel approach that can be applied to multiple functions. Our method applies a communication reduction scheme to the set of functions, rather than to each function independently, keeping a low communication volume. Evaluation on several real-world datasets shows that our method can track many functions with reduced communication, in most cases incurring only a negligible increase in communication over distributed approximation of a single function.

CCS CONCEPTS

•Computing methodologies →Distributed algorithms;

KEYWORDS

distriubted streams, continous approximation, multiple fuinctions

ACM Reference format:
Arnon Lazerson, Moshe Gabel, Daniel Keren, and Assaf Schuster. 2017. One for All and All for One: Simultaneous Approximation of Multiple Functions over Distributed Streams. In *Proceedings of DEBS '17, Barcelona, Spain, June 19-23, 2017,* 12 pages.
DOI: http://dx.doi.org/10.1145/3093742.3093918

1 INTRODUCTION

In the data streaming model, a single site observes a large (possibly infinite) data stream, and keeps an approximation of a predefined function f, using sublinear space [43]. Extensions to this model [3, 15] addressed the problem of approximating multiple functions over a single stream. The inherently distributed nature of the data in many applications (e.g. sensor networks [45] or network operation centers [13]), as well as the size of the data, inspired wide interest in the distributed case, and motivated the continuous distributed monitoring model [12]. This model addresses the common situation where physically distributed sites must keep a continuous approximation of a function applied to the union of their observations.

Previous works on continuous function approximation (tracking) over streams can be roughly categorized as addressing:

SFSS – a single function over a single stream;
MFSS – multiple functions over a single stream;
SFDS – a single function over distributed streams.

The goal of SFSS is to optimize for the space/time performance of approximating a single function. Most work on MFSS focuses on resource sharing of the underlying data structures to achieve faster performance and better space utilization. In SFDS the primary concern is communication cost, since communication is the major energy drain in sensor-networks, and excess communication in a data-network interferes with its normal operation [33].

MFSS approaches have been used to improve multi-function performance, but not in the distributed case, where communication cost is the primary concern. Previous work on distributed approximation (SFDS) developed communication-efficient tracking solutions for many important functions. However, they dealt with the approximation of a single function at a time. Yet, in real distributed systems, we are seldom interested in only one type of function; for example, we may want to simultaneously track both the mean and the variance of a distributed histogram. Previous work did not address the problem of tracking multiple functions in a distributed setting, implicitly assuming that multiple functions must be tracked independently.

We pose a new question: is it possible to simultaneously track a combination of *arbitrary* functions, using less communication than required to track the functions independently? To the best of our knowledge, this important problem has not been addressed before. We introduce a new tracking category called **MFDS** that addresses multiple functions over distributed streams. This is a natural extension of both MFSS and SFDS.

Distributed query processing addresses a similar question in the context of SQL-like queries over aggregates. It is discussed in more detail in Section 2.

Our Contribution

We define the problem of tracking multiple functions over distributed steams, which rises in many real-life scenarios, and show that non-trivial communication reduction is possible. We show

DEBS '17, Barcelona, Spain

DOI: http://dx.doi.org/10.1145/3093742.3093918

that a simple extension of the geometric monitoring framework provides a general approach to the problem of efficient simultaneous approximation of multiple functions over the aggregate of distributed streams. We empirically validate our work on four real-life datasets. Our evaluation addresses complex non-linear, non-monotonic functions, applied to multi-dimensional data. Our simulations show that our approach can be used to approximate multiple functions with nearly the same communication cost of the worst (in terms of communication cost) function, rather than the sum of costs of all functions.

2 RELATED WORK

SFDS approaches – monitoring a single function over distributed streams observed by remote sites while reducing communication – has received a lot of attention, due to increased data rate and volumes.

Sketching approaches [11] provide a compact summary of the data, which is then used to approximate the value of a function. Sketching approaches were originally intended for single stream settings (SFSS), but they were extended to fit the distributed case. Sketching reduces the *size* of each message but not the number of messages. A particular sketch can be used to approximate only a specific function or a specific set of functions [11]. For example, AMS sketches [1] were designed to approximate frequency moments (and join sizes), but not percentiles.

Closer to our work in spirit are SFDS approaches based on local constraints. Carefully constructed constraints are placed at the distributed sites to guarantee the desired approximation. In these approaches the sites communicate only when their local constraints are violated, thus decreasing the *number* of messages. Sketching is commonly combined with these approaches, to reduce message size. [23, 33, 37].

Early work on monitoring distributed streams via local constraints mainly addressed specific functions, initially linear or monotonic functions. Thresholding of linear functions is explored in [31, 33]. Distributed monitoring of monotonic functions was studied in [14, 42]. Non-monotonic functions were discussed in [2]. In [46] non-monotonic functions were addressed by representing them as a difference of monotonic functions. Top-k monitoring was studied in [4]. Distributed HillOut was proposed in [44] to detect outliers in grid systems. In [48] local constraints were placed on one-dimensional variables to monitor the value of a polynomial. Perturbative analysis of eigenvalues was proposed in [28], and was applied to traffic volume data to monitor system health. A theoretical study of the monitoring problem is provided in [14].

These earlier papers constitute a large collection of special-purpose algorithms, each tailored to the type of function of interest (similar to sketches in that sense).

A general approach, *geometric monitoring* (GM), was introduced in [50], initially to monitor threshold crossings using spherical local constraints (covering spheres). The covering spheres method can be used to monitor any arbitrary function of the aggregate of distributed streams. Follow-up work [25] extended the covering spheres method with a prediction model; the more general notion of Safe Zones (SZs) was introduced in [35]. A privacy-preserving variant was proposed in [17]. In [37], GM was extended by the *convex decomposition* (CD) approach, which breaks complex global constraints into an intersection of half-spaces. The convex bound method (CB) [36] directly derives simple local constraints, allowing for highly improved runtime over the covering spheres method. Very recently, [22] proposed a systematic SFDS approach, which constructs a complex safe zone from simpler safe zones.

Distributed multi-query processing is somewhat related to our work. Research in this area is focused on optimizing the communication cost of multiple SQL-like queries over aggregates of distributed streams. The optimization techniques are based on exploiting reuse opportunities (e.g. sharing partial results common to several aggregates or sharing data between overlapping sliding windows) [27, 29, 40]. These techniques reduce the size of the messages sent (like sketches), but not their number, which is the primary goal of our work. Moreover, multi-query processing targets queries and SQL-like aggregates, whereas we are concerned with the approximation of arbitrary general functions.

3 MFDS PROBLEM DEFINITION

Following [12, 14, 23], we consider a system of k remote sites (or nodes), each observing a local stream of events, and a designated coordinator site. Remote sites only communicate with the coordinator, and vice versa; we assume latency is smaller than event arrival rate. The goal is to continuously approximate the value of a predetermined set of functions $\{f_1 \ldots f_\ell\}$, with corresponding error bounds $\{\epsilon_1 \ldots \epsilon_\ell\}$, defined over the aggregate of the local data streams.

Formally, let v^j be the *local vector* of node j, $j \in \{1, .., k\}$, where v^j is incrementally updated by node j as new event arrives. Let the global vector v be the aggregate of current local vectors.

We wish to maintain a set of approximations $\hat{f}_i$ such that:

$$\forall i \in 1 \ldots \ell : f_i(v) - \epsilon_i \leq \hat{f}_i \leq f_i(v) + \epsilon_i \quad ,$$

or, alternatively, relative or multiplicative approximation bounds. We discuss arbitrary functions of the average or weighted mean of the local vector: $v = \frac{1}{k} \sum_{j=1}^{k} v^j$. Many aggregations can be expressed in this form. For example, [19] shows how to express least square regression of the *union* of local matrices to f defined on the average.

The MFDS problem is an extension of the SFDS problem, where $\ell = 1$. The challenge is to approximate all functions f_i while minimizing the number of messages sent between nodes and the coordinator.

In the *naive approach* each node forwards every observation to the coordinator. Since the coordinator knows the true global aggregate, it can compute the values of all functions directly with no loss of accuracy. Moreover, tracking multiple functions bears no extra communication cost relative to the approximation of a single function. In other words, the communication cost for the naive approach is independent of the number of functions. However, the communication cost of the naive method can be prohibitively high. Indeed, this is the entire motivation for SFDS approaches.

The second approach to approximating multiple functions is to *independently* track each one, using communication-efficient SFDS tracking algorithms. SFDS algorithms can substantially reduce communication, and if the sum of total messages is below that of the

naive, then this approach can be effective. This approach is not scalable, however, as the total cost of communication strongly depends on the number of functions: it is the sum of costs for monitoring individual functions. As ℓ grows, performance deteriorates, and the naive approach becomes preferable.

It would be ideal to find an approach that combines the best of both worlds: one that is communication-efficient and independent of (or at least weakly dependent on) the number of functions. However, since each traditional SFDS algorithm is tailored to a particular function (or set of functions), it seems impossible to combine arbitrary algorithms. For example, it is hard to see how one could combine monitoring entropy [2] and quantiles [53] of distributed streams.

4 TRACKING MULTIPLE FUNCTIONS

The root problem with combining most SFDS algorithms is that they share no unifying principle. Combining several algorithms for different functions may be difficult (if not impossible), since the monitored functions may be very different.

Our key observation is that while the outputs of arbitrary functions on the global aggregate may be very different, they all share the same input v. This allows us to take advantage of two key principles of the geometric monitoring (GM) approach. First, the GM approach expresses the constraint on the value of the function $f(v)$ as a constraint in the *domain* of the function rather than a constraint on its image. In other words, we constrain the input vector v, and not the output value $f(v)$. Second, GM works by decomposing the global constraints to convex local constraints: all local constraints, regardless of the monitored function f_i, are convex constraints defined on v^j. The convexity requirement plays a key role in GM (Section 4.1) and in our approach (Section 4.2). These two common properties mean that GM algorithms are *composable*: we can efficiently combine multiple algorithms for arbitrarily different functions in one tracking algorithm.

The rest of this section is organized as follows. Section 4.1 reviews the geometric monitoring framework, and explains how the threshold crossing problem is applied for tracking. Section 4.2 describes our approach, showing how multiple single-function algorithms can be combined to create a multi-function algorithm.

4.1 Geometric Monitoring Framework

Recall that we consider a system of k remote nodes and a coordinator, where node j maintains a local vector v^j, and the monitored function is applied the aggregated vector, defined as $v = \frac{v^1+..+v^k}{k}$. One of the fundamental problems in such a distributed system is the distributed threshold problem: given a function f and a threshold T, we wish to know whether $f(v) \leq T$.

GM was originally designed to address the distributed threshold problem. We next describe how GM is applied to monitor threshold crossings, and then show how to use it as a building block for tracking function values.

The Distributed Threshold Problem. Given a threshold T, we wish to know whether $f(v) \leq T$. GM rests on a geometric interpretation of this problem. Define the *admissible region* $A \triangleq \{u \mid f(u) \leq T\}$, and ask the equivalent question: is $v \in A$?

Protocol description. Let v^j be the current local vector at node j, and let v be the current global vector . Let v_0^j, v_0 be the values of v^j and v time $t = 0$, respectively. At time $t = 0$, the coordinator performs a synchronization step: it collects the local vectors $\{v_0^j\}$ from the nodes, computes the *estimate vector* $v_0 = \frac{v_0^1+..+v_0^k}{k}$, and shares it with the nodes. Note that the estimate is simply v at $t = 0$. As time passes, the local vectors $\{v^j\}$ change, and their values drift away from their initial values $\{v_0^j\}$. Define the the *drift vector* at node j as $\delta^j = v^j - v_0^j$. The following property always holds [50]:

$$v = \frac{v^1 + \cdots + v^k}{k} = v_0 + \frac{\delta^1 + \cdots + \delta^k}{k} \quad .$$

Let the *safe zone*, SZ, be a *convex* subset of A (containing v_0). The convexity of SZ guarantees that if for all j, $w^j \triangleq v_0 + \delta^j \in SZ$, then $v \in SZ$. Since $SZ \subseteq A$, then $f(v) \leq T$. Each node can independently check whether $w^j \in SZ$, as both v_0 and δ^j are computed from local data at the node. As long as all these local constraints hold, the global constraint ($f(v) \leq T$) also holds. Upon every new observation, node j updates δ^j and checks whether its local constraint $w^j \in SZ$ holds. If the constraint is violated, the node notifies the coordinator, which resolves the violation by repeating the synchronization step.

Observe that a set S can be used as a safe zone if and only if it has the three following properties: (1) S is convex ; (2) it is contained in the admissible region: $S \subseteq A$; and (3) $v_0 \in S$. Properties 1 and 2 guarantee that v is in the admissible region. To avoid a violation at $t = 0$, property 3 must be maintained.

Ideally, a "good" safe zone would contain not only v_0, but a large neighborhood around it, to reduce local violations. The geometric monitoring literature offers several methods for finding a "good" safe zone, given v_0, a function and a threshold [36, 37, 50, 51]. There are also safe zones designed for specific functions [19].

From Threshold Crossing to Tracking. Consider the task of approximating (at the coordinator) the value of a function f over the global vector $v = \frac{\sum v^j}{k}$, within ϵ absolute error (relative and multiplicative errors are handled in a similar manner). In other words, the coordinator must keep an approximation $\hat{f}$, such that $\hat{f} - \epsilon \leq f(v) \leq \hat{f} + \epsilon$.

As in the distributed threshold problem, at time $t = 0$, the coordinator collects the data from all the nodes and calculates the estimate v_0, and the approximation $\hat{f} \triangleq f(v_0)$. Then, the coordinator shares v_0 with the nodes. Each node calculates $f(v_0)$ and sets two thresholds: $f(v_0) - \epsilon$ from below and $f(v_0) + \epsilon$ from above (alternatively, we can set multiplicative or relative bounds). Note that the estimate vector defines the admissible region $A = \{u \mid f(v_0) - \epsilon \leq f(u) \leq f(v_0) + \epsilon\}$, and that $v_0 \in A$.

Having reduced approximation to a threshold crossing problem, we continue as before: nodes monitor local constraints and the coordinator resolves any violations. This protocol guarantees that $f(v_0)-\epsilon \leq f(v) \leq f(v_0)+\epsilon$, at all times. Algorithms 1 and 2, outline the protocols of the coordinator and a single node, respectively. For simplicity, we describe the simplest violation recovery protocol, known as *eager* synchronization [23, 50].

Algorithm 1 Synchronization (Coordinator)

1: Poll all nodes for v^j
2: Calculate the estimate $v_0 \leftarrow \frac{\sum v^j}{k}$
3: Distribute v_0 to all nodes

Algorithm 2 Node j is updated with new observation

1: Update the local vector v^j
2: Compute the drift: $\delta^j \leftarrow v^j - v_0^j$ and $w^j \leftarrow v_0 + \delta^j$
3: satisfied $\leftarrow w^j \in SZ$
4: **if not** satisfied **then**
5: Notify the coordinator to trigger synchronization
6: Wait for new v_0 from the coordinator
7: Calculate $f(v_0)$
8: Set new bounds $f(v_0) - \epsilon$ and $f(v_0) + \epsilon$
9: Compute the respective safe zone SZ
10: Set $v_0^j \leftarrow v^j$

4.2 The Common Admissible Region

The geometric monitoring framework is based on the notion of the admissible region. Using GM to track multiple functions requires multiple admissible regions.

Our approach is based on a simple, yet powerful observation: given a set of ℓ functions $\{f_1 \dots f_\ell\}$ with corresponding error bounds $\{\epsilon_1 \dots \epsilon_\ell\}$, the common admissible region CA for the entire set is the intersection of individual admissible regions:

$$CA \triangleq \{v \mid v \in A_i, i = 1 \dots \ell\} = \bigcap_{i=1}^{\ell} A_i \quad ,$$

where A_i is the admissible region for (f_i, ϵ_i). Note that CA is not empty, and must contain at least v_0, since $v_0 \in A_i$ for all A_i. As in Section 4.1, applying each function f_i to the common estimate vector v_0 yields its current approximated value: $f_i(v_0)$.

Figure 1 shows an example of a common admissible region CA, which is the intersection of A_f and A_g. A_f and A_g are defined by $f(x, y) = y - x^2 - 0.8$ and $g(x, y) = x^2 + y^2 - 1$ with absolute approximation bounds $\epsilon_f = 1$ and $\epsilon_g = 0.5$, respectively.

Having defined the admissible region CA we can apply one of the existing methods to find a good safe zone within it.

Direct Optimization. One method for finding a safe zone is presented in [35]. Given the admissible region A and v_0, this approach searches for "good" convex subset C, by solving the following optimization problem:

$$\text{maximize} \int_C p(x)d(x) \text{ , such that } C \text{ is convex, } C \subseteq CA, v_0 \in C,$$

where $p(x)$ is the probability distribution of the monitored data.

While in theory this approach can be used over the common admissible region, in practice, optimizing over all convex the subsets is impossible [35]. Even if we limit ourselves to convex polygons, the optimization problem is nontrivial, especially if the data is high dimensional [35]. Worse, CA is the intersection of multiple admissible regions, which increases the difficulty of the optimization problem.

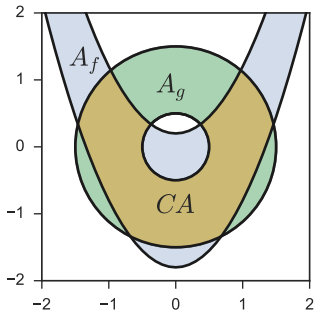


Figure 1: The common admissible region CA (yellow) is the intersection of two admissible regions A_f (blue) and A_g (green). As long as $(x, y) \in CA$, both f and g maintain their respective approximation bounds.

Covering Spheres. The covering spheres method [35, 50] relies on the bounding lemma [50]: Let $B(p, q)$ be the sphere whose diameter is $\overline{pq}$. If for every node j $B(v_0, v_0 + \delta^j) \subseteq A$, then $v \in A$. As before, each node checks its local constraint: $B(v_0, v_0 + \delta^j) \subseteq A$. The equivalent safe zone is the set of all vectors u, such that the sphere $B(u, v_0)$ is fully contained in the admissible region.

We can apply the covering spheres method directly to the common admissible region by extending the local constraint so that the sphere must be included in all individual admissible regions A_i. In other words, each node j now checks whether $B(v_0, v_0 + \delta^j) \subseteq A_i$ for all i.

While this strategy may be useful in some cases, it suffers from two drawbacks. First, other methods such as convex decomposition (CD) [37] and convex bound (CB) [36] are known to achieve better results in terms of communication. More importantly, covering spheres suffers from a prohibitively long runtime in many cases, since checking whether a sphere is entirely inside the admissible region can be very difficult [36]. With multiple admissible regions, the problem becomes even worse.

Common Safe Zone (CSZ). To avoid the complications of dealing with the common admissible region directly, we suggest an alternative approach that is simpler to apply in practice. First we find a safe zone for each constraint separately, and then intersect the safe zones. Formally, the common safe zone is given by:

$$CSZ \triangleq \bigcap_{i=1}^{\ell} SZ_i \quad ,$$

where SZ_i is the safe zone for (f_i, ϵ_i).

Observe that the CSZ is indeed a legal safe zone for the combined constraints, that is, it fulfills the three required properties of a safe zone (Sec 4.1). First, CSZ is convex, because it is an intersection of convex sets. Second, $CSZ \subseteq CA$, since the intersection of the admissible regions contains the intersection of safe zones (each safe zone is contained in its respective admissible region). Third, v_0 is contained in CSZ, because v_0 is contained in each individual safe zone. Moreover, good safe zones include a large neighborhood of v_0, therefore, it is likely that their intersection also includes a neighborhood of v_0; namely, CSZ, in some sense, is also a "good"

Algorithm 3 Node j is updated with new observation

```
1: Update the local vector v^j
2: Compute the drift: δ^j ← v^j − v_0^j and w^j ← v_0 + δ^j
3: satisfied ← ⋀_{i=1}^{ℓ} w^j ∈ SZ_i
4: if not satisfied then
5:     Notify the coordinator to trigger synchronization
6:     Wait for new v_0 from the coordinator
7:     for i in 1 . . . ℓ do
8:         Calculate f_i(v_0)
9:         Set new bounds f_i(v_0) − ϵ and f_i(v_0) + ϵ
10:        Compute respective safe zone SZ_i
11:    Set v_0^j ← v^j
```

safe zone. Note that since v_0 is contained in all individual safe zones, we can compute the approximated values by applying the functions to v_0: $f_1(v_0) \dots f_\ell(v_0)$.

Rather than explicitly computing the common safe zone, nodes can simply test whether the vector is inside all of the individual safe zones. The *CSZ* method requires a small modification to the algorithm of the node (Algorithm 2). In the new algorithm, Algorithm 3, node j must check whether v^j is included in the safe zones of all functions. The algorithm of the coordinator is unchanged.

The *CSZ* method is applicable to multiple functions with different approximation error bounds. It can handle both relative and absolute error bounds as part of the same tracking algorithm, and different variants of GM can be combined. Moreover, GM framework extensions naturally complement our method. For example: *lazy violation resolution* [50], where the polling of all nodes in Algorithm 1 is replaced with more efficient, incremental polling; *local violation resolution* [34] where disjoint node pairs are locally balanced; and *reference point prediction* [25, 26], which predicts the current value of v from past values of v_0 to avoid synchronizations.

Example. Consider a simple scenario in which we track $f(x, y) = x^2 - y - 0.4$ and $g(x, y) = y^2 - x$, within an absolute approximation bound of $\epsilon = 1$. That is, we want to maintain an estimate $v_0 = (x_0, y_0)$, such that $f(x_0, y_0) - 1 \le f(x, y) \le f(x_0, y_0) + 1$ and $g(x_0, y_0) - 1 \le g(x, y) \le g(x_0, y_0) + 1$.

Figures 2a and 2b illustrate the admissible regions of f and g and the respective safe zones induced by CB [36] for $v_0 = (x_0, y_0) = (0, 0)$. While the functions and their respective admissible regions and safe zones are very simple, the shape of the common admissible is quite complicated (Figure 2c). Finding a "good" safe zone in this admissible region is not trivial, since we need to find a large convex neighborhood of v_0. The common safe zone induced by the *CSZ* method is shown in Figure 2d. This is a "good" safe zone, in the sense that it cannot be expanded without violating convexity.

5 EVALUATION

Geometric monitoring has previously been applied to many problems, but always in an SFDS setting. Applications include efficient outlier detection in sensor networks [9]; variance monitoring and outlier detection for multivariate time series [18, 21, 49]; sketch-based monitoring of norm, range-aggregate, and join-aggregate queries over distributed streams [23, 37]; monitoring least-squares regression models [19]; monitoring the prediction error of a global model [30]; inducing decision trees in a large distributed network [5]; distributed entropy approximation [20]; and monitoring highly distributed data streams using sample based techniques [24].

In this section we present and evaluate four applications requiring the approximation of multiple functions: regression and condition number; top-k PCA scores; distance metrics; and five-number summary. These applications utilize both existing and newly derived safe zones. The functions used by the applications are non-linear and non-monotonic, and are applied to multi-dimensional data. We also demonstrate tracking a complicated function by decomposing it to simpler primitives, and tracking those. Additionally, we compare the CSZ and covering spheres methods.

Our evaluation shows that in most cases the CSZ method incurs almost the same communication as tracking a single function in each application, and is always below the communication required to independently track all functions. Finally, we demonstrate that our approach scales with the number of nodes.

5.1 Evaluation Methodology

We applied the *CSZ* tracking method to the four applications listed above, and compared its cost to independently approximating each function ("indep"). Four different real-life datasets were used in the evaluation process. We assumed a sliding window scenario: for a sliding window of size w, the local vectors are computed from the last w records.

We simulated the tracking protocol for k streams: at each simulation step, every node observes a new event. We counted the number of messages, including synchronization overhead, sent during the simulation; latency is assumed to be smaller than event arrival rate. Our main cost metric is the *ratio to naive*: the number of messages divided by the number of messages sent by the naive method, in which each node forwards every observation to the coordinator.

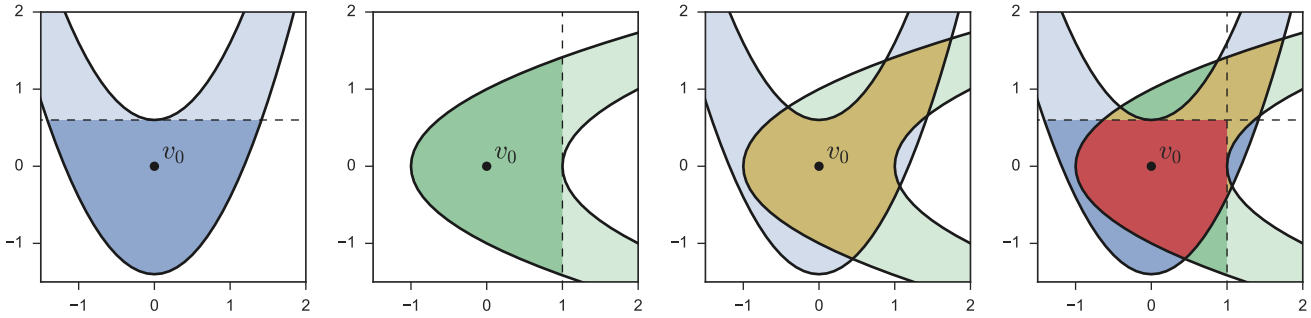
Our main criterion for success is reducing the cost below that of the independent method; in other words, we want the *ratio to independent* metric to be below 1.0. In the best case, the *CSZ* method would cost the same as the maximum cost of tracking the independent functions. Note that the *CSZ* method can never do worse than the sum of independent methods.

5.2 Data Sets

We used four datasets: Intel Lab Data (ILD) [7], the Reuters Corpus (REU) [38], a Twitter crawl (TWIT) [39], and the 10 percent sample supplied as part of KDD Cup 1999 Data (KC)[32]. For KC, REU, and TWIT, we used round-robin order to assign observation to the k simulated nodes, with $k = 10$ by default.

ILD contains data collected from 54 sensors deployed in the Intel Berkeley Research lab between February 28th and April 5th, 2004. The sensors collected humidity, temperature, light and voltage values once roughly every 30 seconds. Since some sensors have a lot of missing and/or out of range values, we selected 16 sensors with (relatively) high quality data[1]. We used data from nine days (March 1–9, 2004). The data was normalized to zero-mean and unit

[1] Sensors 24, 25, 27, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 40, 41, and 42.



(a) Admissable region (light blue), and safe zone (dark blue) for f.

(b) Admissable region (light green), and safe zone (dark green) for g.

(c) Common admissable for f and g (yellow).

(d) Common safe zone for f and g (red).

Figure 2: While the shapes of the admissible regions of f and g and their safe zones are simple (Figures (a), (b)), the shape of the common admissible region (Figure (c)) is quite complicated. The CSZ method easily finds a good common safe zone (Figure (d)) by intersecting the individual safe zones.

variance and re-sampled to 60 second intervals. Missing values were filled in using linear interpolation.

KC was used in the "Third International Knowledge Discovery and Data Mining Tools Competition" for the task of intrusion detection. It contains information about TCP connections described by 41 features, such as duration, protocol, bytes sent, or bytes received. Out of these we extracted the 37 real-valued attributes. Our experiments were performed on the top 50K records of the normalized (zero mean, unit variance) data.

REU (RCV1-v2 collection) consists of 804,414 categorized news stories reported by Reuters between August 20, 1996, and August 19, 1997. Each document is represented as a feature vector of length 47,236. We used the top 2050 features left after removing features that appear in less than 1% of the documents.

TWIT (Dataset-UDI-TwitterCrawl-Aug2012) was collected in May 2011 and contains 50 million tweets. It was originally collected for profiling users and relationships in a social network. We filtered the dataset to obtain only hashtagged tweets, which left us with 9 million tweets from 140,000 users. We used the NLTK [6] package to tokenize and stem the text of the tweets in TWIT, and then selected the top 1250 features, ignoring features appearing in less than 0.1% of the tweets.

5.3 Regression and Condition Number

Linear regression fits a linear model to a set of observations, and is commonly used for prediction and analysis tasks. Some linear systems are very sensitive to measurement errors, meaning solutions may be inaccurate. The condition number is a common metric to assess the stability of a linear system. Therefore, we are interested in tracking both a least squares regression model and its condition number.

Regression. Assume the data consists of n observations, where observation i is composed of a vector of m predictors x_i and a response variable (scalar) y_i. Let X be the $n \times m$ matrix of predictor row vectors $X \triangleq (x_1^T, ..., x_n^T)^T$, and let and y be the column vectors of the response variables $y \triangleq (y_1, ..., y_n)^T$. The Generalized Least Squares (GLS) regression model is a linear transformation $\beta \in \mathbb{R}^m$ that minimizes the Mahalanobis distance $(y - X\beta)^T S^{-1}(y - X\beta)$. The optimal solution to GLS is given by:

$$\beta = (X^T S^{-1} X)^{-1} X^T S^{-1} y \quad .$$

where S is error correlation matrix [47]. GLS is designed to handle correlated measurements and is very useful in time-series analysis. Setting $S = I$ yields $\beta = (X^T X)^{-1} X^T y$, the solution to the Ordinary Least Squares (OLS) model $y = X\beta$.

Continuous tracking of a regression model was studied in [19]. Given a sliding window of n observations, we aim to maintain a continuous approximation β_0 of the current GLS model β, such that $||\beta_0 - \beta|| \leq \epsilon$, for some user defined ϵ.

The local vector at node j is defined as the pair $v^j = (A^j, c^j)$, where

$$A^j = \sum_{i=1}^{n} S^{-1} x_i x_i^T \ , \ c^j = \sum_{i=1}^{n} x_i S^{-1} y_i \quad ,$$

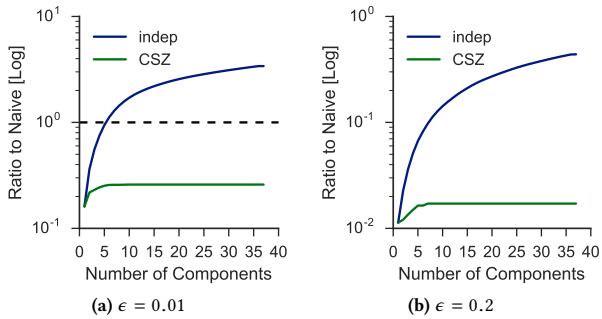
and $\{(x_i, y_i)\}$ are the observations pairs at the node. The global vector $v = (A, c)$ is the average of the local vectors $\{v^j\}$. The estimate vector $v_0 = (A_0, c_0)$ is v at $t = 0$ as usual. The drift is given by (Δ^j, δ^j), where $\Delta^j = A^j - A_0^j$ and $\delta^j = c^j - c_0^j$. The model at $t = 0$ is defined by $\beta_0 = {A_0}^{-1} c_0$. The safe zone is described in Appendix A.1.

Condition Number. The condition number κ is the ratio between the largest to smallest singular values [41]. For the symmetric scatter matrix $A = X^T S^{-1} X$ used in GLS, it is the ratio between maximal eigenvalue λ_1 and the minimal eigenvalue λ_m: $\kappa = \frac{\lambda_1(A)}{\lambda_m(A)}$.

The task is to track the condition number of A with ϵ relative error: $(1 - \epsilon)\kappa_0 \leq \kappa \leq (1 + \epsilon)\kappa_0$, where κ_0 is the approximation and κ is the current true value of the condition number. The safe zone of the condition number is described in Appendix A.2

Empirical Evaluation. We simulated the tracking algorithms using the ILD dataset, with a sliding-window of size 600. Our goal here was to track a global regression model where the predictors x are temperature, light, and voltage, and the response variable

Figure 3: Regression and condition number communication costs relative to the naive method. The cost of the CSZ method is nearly the same as the worst of the two.



(a) $\epsilon = 0.01$ **(b)** $\epsilon = 0.2$

Figure 4: Communication cost for tracking increasing number of PCA scores (log scaled) for strict and relaxed error bounds. Independent tracking scales poorly: in both cases, the communication cost of indep grows sharply. For strict $\epsilon = 0.01$ indep quickly becomes worse than naive. The CSZ method scales much better: its cost grows slowly and becomes constant after the first few scores. In both cases, the cost of CSZ remains well below both naive and indep.

y is humidity. Following [19], we used a Toeplitz autocorrelation matrix S, such that the values on diagonal i are 0.98^i.

Figure 3 shows the relative communication cost of tracking the regression model and condition number for different values of model approximation error ϵ. The condition number was allowed a relative error of 0.2 in all cases. The communication cost of the *CSZ* method is only 5–15% higher than the costliest independent function (regression for $\epsilon \leq 0.3$ and condition number for $\epsilon \geq 0.4$), regardless of model error . The *CSZ* method requires about 60% of the communication volume required by the sum of independent methods.

5.4 PCA-Score

PCA (Principal Component Analysis) is a popular dimension reduction technique with numerous applications. Given a set of vectors, PCA finds a low-dimensional approximation of the vectors.

The PCA-Score or "proportion of the total variance explained" t_k is a simple and popular stopping rule [10], measuring the cumulative variance explained by the first k principal components. To compute t_k given a scatter matrix $A_{m\times m}$, first compute its eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$ (each eigenvalue represents the variation explained by the associated principal component). Then divide the sum of the top-k eigenvalues by the sum of all eigenvalues:

$$t_k = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{m} \lambda_i} \quad .$$

The top ℓ PCA-Score's $t_1 \ldots t_\ell$ can be used to construct a screeplot [10], or find how many dimensions must be retained to guarantee a specific percentage of the variance. Furthermore, using $t_i - t_{i-1}$ the contribution of a specific eigenvalue λ_i can be approximated. This has applications in graph theory when A is a adjacency matrix or graph Laplacian [8, 52].

Our objective is to approximate t_k (for multiple values of k) with an additive approximation bound ϵ: $t_{k,0} - \epsilon \leq t_k \leq t_{k,0} + \epsilon$, where t_k is the current PCA-Score and $t_{k,0}$ is the PCA Score at time $t = 0$, as always.

Given a sliding window of n observations $\{xi\} \in \mathbb{R}^m$, the local vector at node j is the scatter matrix:

$$A^j = \sum_{i=1}^{n} x_i x_i^T \quad .$$

The global vector $v = A$ is the average of the local vectors, and the estimate vector is $v_0 = A_0$ (v at time $t = 0$). $v_0^j = A_0^j$ is the local vector at $t = 0$, $\Delta^j = A^j - A_0^j$. The safe zone of the PCA score is described in Appendix A.3

Empirical Evaluation. We simulated our algorithm using the KC dataset with a sliding window of size 1000 in each node.

Figure 4 illustrates how communication costs of *CSZ* and indep scale when varying the number of tracked scores ℓ from 1 to 37 (the data has 37 dimensions). Figure 4a shows communication costs with an absolute error bound $\epsilon = 0.01$. The cost of indep grows above that of the naive method after only 6 components, eventually reaching over 3 times that of naive. Conversely, the cost of the *CSZ* method reaches its maximum cost of about 25% of the naive method after 10 components. Figure 4b shows costs for a more relaxed additive error bound, $\epsilon = 0.2$. The cost of indep grows almost linearly, reaching 40% of the naive method for monitoring the full PCA spectrum, while *CSZ* reaches the maximum 2% of naive after 7 components. In both cases, *CSZ* communication is substantially lower than the indep cost (13 times and 25 times, respectively). Additionally, monitoring additional components beyond the first few is practically free with the *CSZ* method.

Figure 5 shows the cost for tracking each of the top $\ell = 10$ PCA scores and the cost of the *CSZ* method, divided by the sum of independent costs (i.e., ratio to independent). All functions $t_1 \ldots t_{10}$ have the same additive error bounds ϵ, and we show communication costs for different values of ϵ. The cost of the *CSZ* method is at most 15% of the total cost of independently monitoring each score, and is only 9–32% higher than the maximum of independent costs in each case. We observe that the costliest single component is different for different error bounds, meaning that tracking a single component is insufficient to guarantee all approximation bounds.

Figure 6 demonstrates the tradeoff between accuracy ϵ and communication when tracking the top 10 PCA scores. Both indep and CSZ offer similar tradeoffs, but CSZ can still maintain reasonable communication reduction even for strict approximation accuracy.

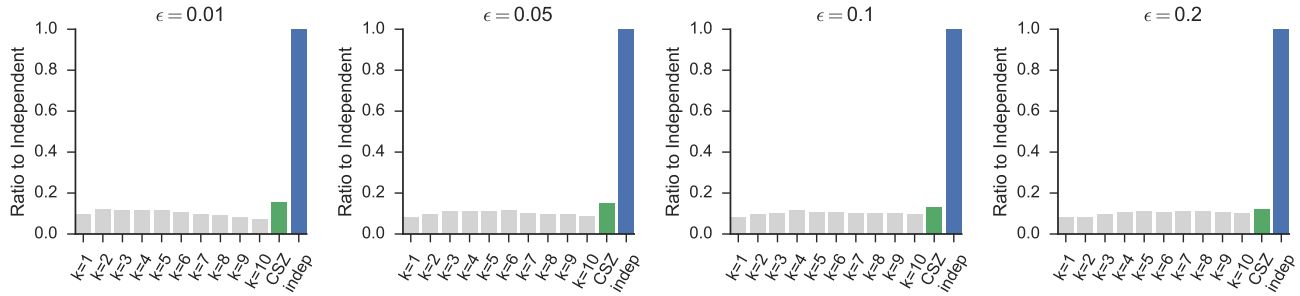


Figure 5: PCA-Score communication relative to the independent combination (the sum of individual communication costs). The CSZ method requires only slightly more communication than tracking the worst individual function, and reduces communication by a factor of 7 relative to indep.

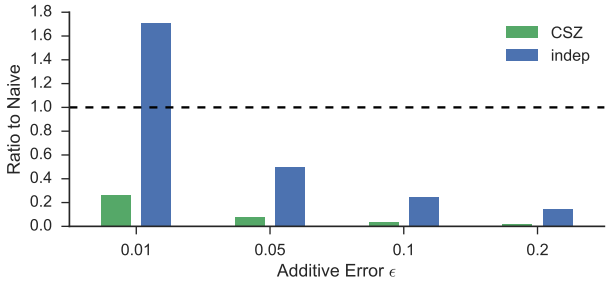


Figure 6: Communication cost for top 10 PCA scores relative to the naive method. For $\epsilon = 0.01$ indep is actually worse than the naive method, while CSZ is better by a factor of 4.

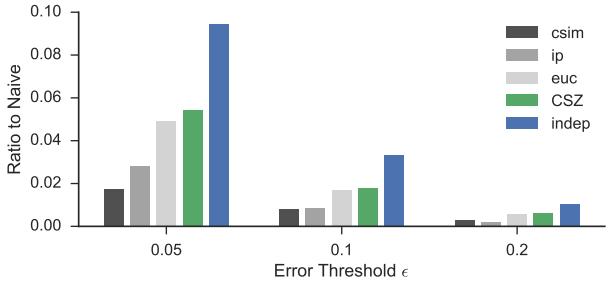


Figure 7: Communication costs of distance metrics relative to the naive method. The communication incurred by CSZ is roughly the same as communication cost for tracking euc.

5.5 Distance Metrics

Distance metrics (and similarity measures) are key to many data-mining tasks such as information retrieval, clustering, and classification. Different tasks require different metrics, possibly at different accuracy levels.

Given two n-dimensional vectors x and y, we simultaneously track three important distance metrics: inner product (ip), cosine similarity (csim), and Euclidean distance (euc). In other words, we monitor three different similarity functions over the same pair of vectors (x, y): the inner product $f_{\text{ip}}(x, y) = \langle x, y \rangle$, the cosine similarity $f_{\text{csim}}(x, y) = \frac{\langle x, y \rangle}{\|x\| \|y\|}$, and the Euclidean distance $f_{\text{euc}}(x, y) = \|x - y\|$.

The local vector is the pair $v^j = (x^j, y^j)$, where x^j and y^j are computed from the current sliding window of node j. As usual, the global vector v is the average of the local vectors. The estimate vector $v_0 = (x_0, y_0)$ equals v at $t = 0$. v_0^j is the value of the local vector at $t = 0$, and $\delta^j = v^j - v_0^j$. We approximate f within a relative error of ϵ: $(1-\epsilon)f(x_0, y_0) \le f(x, y) \le (1+\epsilon)f(x_0, y_0)$. The safe zones of the distance metrics are described in Appendix A.4.

Empirical Evaluation. We tracked the three distance metrics on REU with relative errors $\epsilon \in 0.05, 0.1, 0.2$ and a sliding window of the last 1,000 documents. Following [23], each document is assigned as vector x or y according to its category. Figure 7 displays the relative communication cost results for the different values of ϵ. The cost of *CSZ* is only 4–10% higher than that of euc which is the costliest of the three distance metrics (for all error bounds).

5.6 Five Number Summary

Given a vector $v = \{v_1 \dots v_n\}$ the p percentile $f_p(x)$ is v_p, such that p% of $\{v_i\}$ are below v_p.

We wish to track the p-percentile parametric family $\{f_{p_1} \dots f_{p_\ell}\}$. One well known family is Tukey's five-number summary. It consists of the five important percentiles: the minimum ($p0$), the first quartile ($p25$), the median ($p50$), the upper quartile ($p75$), and the maximum ($p100$). Another example of such a family is $p90$, $p95$, $p99$, $p99.9$, common for characterizing latency in web services.

In our evaluation we track the five-number summary $f_{p0}, f_{p25} \dots f_{p100}$ of a (distributed) vector with absolute error bounds $\epsilon_{p0}, \epsilon_{p25} \dots \epsilon_{p100}$: $f_p(v_0) - \epsilon_p \le f_p(v) \le f_p(v_0) + \epsilon_p$.

The local vector v^j is computed from the sliding window at node j, v_0^j is the value v^j at time $t = 0$, and $\delta^j = v^j - v_0^j$. The global vector v is the average of the local vectors, and the estimate vector v_0 equals v at time $t = 0$. The safe zone for the percentile function is given in Appendix A.

Empirical Evaluation. We applied the five-number summary to REU and TWIT datasets with additive error bounds. We used a sliding window of the last 6,700 documents for REU, and the last 1,000 tweets for TWIT. The local vector at each node is composed of the feature counts.

Feature counts are distributed according to a power law: most features have very low counts, while high counts are rare. In other words, low percentiles change rarely, while high percentiles vary widely and often. Hence, we use non-uniform approximation

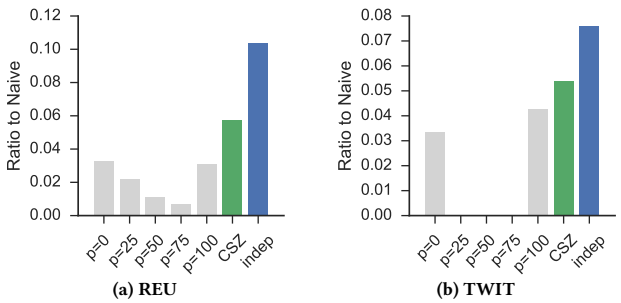


Figure 8: Communication cost for the five-number summary. In both cases the CSZ method is considerably better than independently tracking all percentiles.

bounds: $\epsilon_{p0} = 1$, $\epsilon_{p25} = 2$, $\epsilon_{p50} = 4$, $\epsilon_{p75} = 8$, $\epsilon_{p100} = 16$ in REU, and $\epsilon_{p0} = 2$, $\epsilon_{p25} = 3$, $\epsilon_{p50} = 4$, $\epsilon_{p75} = 5$, $\epsilon_{p100} = 6$ in TWIT (where the window is smaller and counts are lower).

Figure 8 shows communication cost for tracking the different percentiles, as well as the costs of *CSZ* and indep. The total communication cost is dominated by $p = 0$ (min) and $p = 100$ (max). In both datasets the *CSZ* method requires less communication than the independent method. In REU, the *CSZ* method achieves 45% reduction in communication compared to independently tracking all percentiles. Unlike previous cases, however, *CSZ* is more expensive than the single costliest independent function (76% more communication than $p0$). In TWIT, the *CSZ* method reduces communication to only 70% of indep. In this data set, however, the communication cost of costliest single percentile, $p100$, is 56% of indep on its own.

5.7 Function Decomposition

Given a single function to track, we would ideally derive a specialized safe zone for the function, then use GM directly. However, deriving a safe zone for a complicated function may require non-trivial technical work, while using the more general methods (covering spheres or direct optimization) may be impractical (Section 4.1).

We demonstrate a different approach: decomposing the function into simpler primitives and tracking those simultaneously. In other words, we use multiple approximations to provide an approximation for a single, more complicated function. The downside of this decompositional approach is possible loss of communication efficiency compared to the direct approach: a specialized safe zone for the original function is likely to be more optimal than the safe zone created by the CSZ method.

Consider the cosine similarity function $f(x, y) = \frac{\langle x, y \rangle}{\|x\| \|y\|}$, for example. It is neither convex nor concave, and its safe zone is quite complicated (Section A.4) and requires non-trivial application of the CB method [36]. We can decompose cosine similarity into three simpler primitives, namely the inner product $\langle x, y \rangle$ and the norms of x and y $\|x\|$, $\|y\|$, which have much simpler safe zones that are easier to derive (Sections A.4 and A.6). Instead of tracking the cosine similarity directly we can track these components.

Given a relative approximation bound ϵ_{csim} for the cosine similarity function, we must allocate appropriate bounds for the three components ϵ_{ip}, $\epsilon_{\|x\|}$, and $\epsilon_{\|y\|}$ such that: $\frac{1+\epsilon_{\text{ip}}}{(1-\epsilon_{\|x\|})(1-\epsilon_{\|y\|})} = 1 + \epsilon_{\text{csim}}$.

Since individual bounds ϵ_{ip}, $\epsilon_{\|x\|}$, and $\epsilon_{\|y\|}$ must be smaller than the direct bound ϵ_{csim}, we expect that the decomposition will result in more communication.

Empirical Evaluation. How much do we lose by using the decompositional approach rather than directly deriving a specialized safe zone? We compare tracking the components of cosine similarity (inner product and the two norms) using CSZ to tracking the cosine similarity directly using the safe zone derived in [36]. We used the TWIT dataset with a sliding window of size 1000. For simplicity, we assign both norm components the same approximation error $\epsilon_{\|x\|} = \epsilon_{\|y\|} = \epsilon_{\text{norm}}$. We fix the error bound for cosine similarity $\epsilon_{\text{csim}} = 0.1$, and vary the ratio $\alpha = \frac{\epsilon_{\text{norm}}}{\epsilon_{\text{ip}}}$.

Figure 9 shows the communication results for several values of α. Though both CSZ and the direct method are better than independently tracking all components (13% and 45% of indep communication, respectively), the direct method is better than *CSZ*, and in fact incurs less communication than each component individually. The communication cost of the *CSZ* method is at most 3–24% above the worst of the individual components, and nearly an order of magnitude better than the naive method.

We also observe that the allocation of approximation bounds (α) affects the communication cost of the *CSZ* method. The optimal allocation depends on the data and the monitored functions; we leave this problem for future work.

5.8 Covering Spheres

The covering spheres method (Section 4.2) monitors the common admissible region directly. Figure 10 compares it to CSZ using the TWIT and REU datasets (with a sliding window of size 1000), when tracking f_{ip}, the inner product of x and y, and f_{norm}, the norm of the entire vector (the concatenation of x and y), with relative error $1 \pm \epsilon$. Solid bars represent CSZ as before, while bars represent covering spheres (COV). We also show the cost of tracking individual functions using the derived safe zone (for CSZ) or covering spheres with the individual admissible region (COV). We specifically selected f_{ip} and f_{norm} since they can be tracked with COV in reasonable time, unlike the other functions we discuss. For example, f_{csim} with COV is 10^6 times slower than with CSZ [36].

Communication costs for both COV and CSZ are better than indep: their cost is similar to tracking the worst respective individual function. As discussed in Section 4.2, COV sometimes requires more communication than the CSZ method, since using the specialized safe zone for inner product (Appendix A.4) is superior to using the generic covering spheres approach, as shown in [36]. On the other hand, when communication costs are dominated by f_{norm}, COV is equivalent to CSZ, because the safe zone induced by COV is identical to the one derived by CD.

5.9 Scalability With Number of Nodes

The *CSZ* method is scalable, in the sense that its advantage over the independent method does not depend in any way on the number of nodes k. The increase in communication cost of the *CSZ* method is the result of the increased cost of the worst individual function.

We demonstrate this by tracking f_{csim} and f_{ip} using the TWIT dataset with up to 1000 simulated nodes. Figure 11 show that

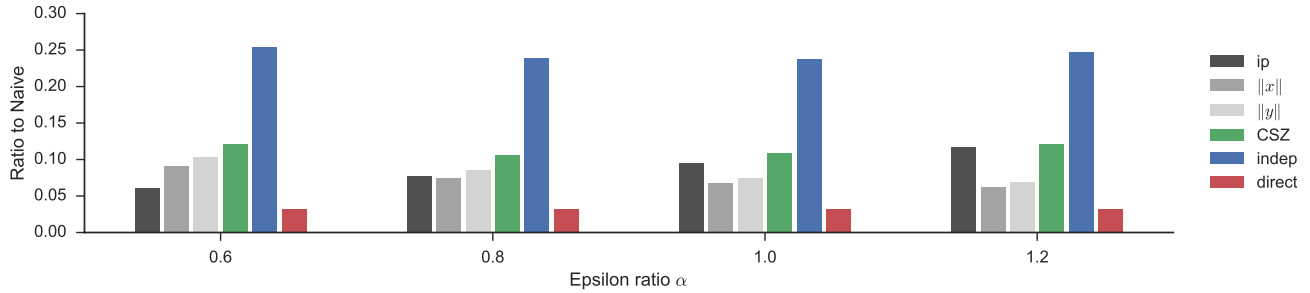
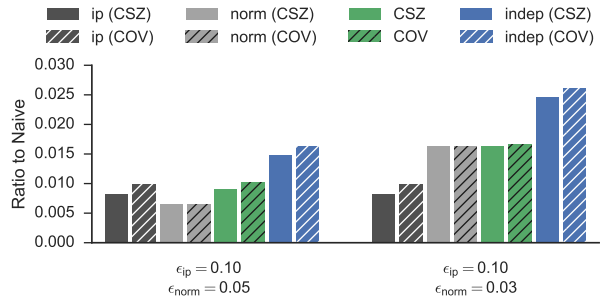
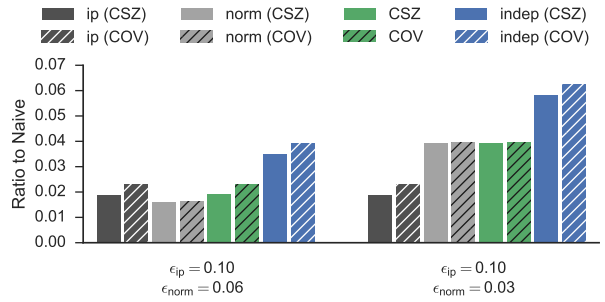


Figure 9: Communication cost for tracking cosine similarity directly and via decomposition to three simpler components, with $\epsilon_{\text{csim}} = 0.1$, and different values for the ratio $\alpha = \frac{\epsilon_{\text{norm}}}{\epsilon_{\text{ip}}}$. The CSZ method costs roughly as much as the worst single component. Tracking the cosine similarity function directly is more efficient still, due to the use of a specialized safe zone.

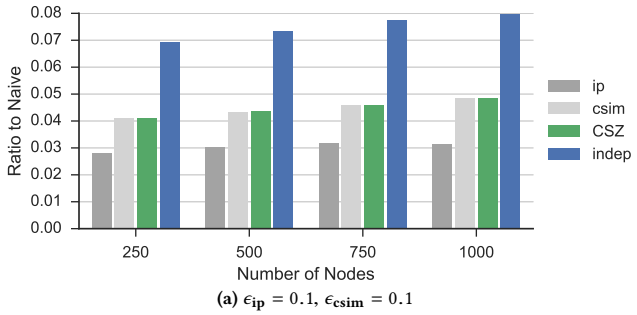


(a) REU

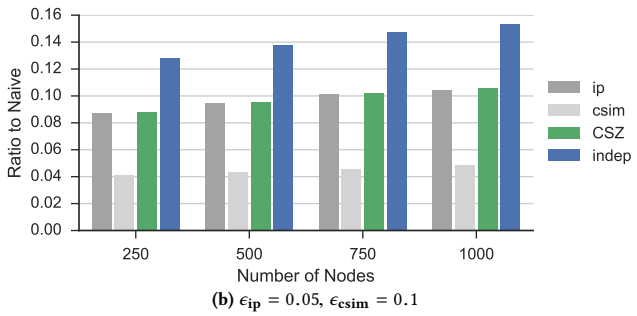


(b) TWIT

Figure 10: Communication costs for COV and CSZ on the REU (a) and TWIT (b) datasets. COV requires more communication than CSZ. For both, cost is similar to tracking the worst respective single function.



(a) $\epsilon_{\text{ip}} = 0.1$, $\epsilon_{\text{csim}} = 0.1$



(b) $\epsilon_{\text{ip}} = 0.05$, $\epsilon_{\text{csim}} = 0.1$

Figure 11: Scalability of the CSZ method with up to 1,000 nodes when tracking cosine similarity (csim) and inner-product (ip). The advantage of CSZ does not depend on the number of nodes. The cost of CSZ is almost similar to that of the worst function in both cases: csim in (a), and ip in (b).

the cost of the *CSZ* method is nearly the same as that of the worst function. Moreover, the difference between the two does not change as k grows. As Figure 11a shows, with $\epsilon_{\text{ip}} = \epsilon_{\text{csim}} = 0.1$ csim dominates the communication cost, and the cost of the *CSZ* method is almost identical. Figure 11b shows that with $\epsilon_{\text{ip}} = 0.05$, $\epsilon_{\text{csim}} = 0.1$, the *CSZ* method matches ip – the costliest function in this configuration.

For simplicity, both indep and CSZ use the eager synchronization protocol described in Algorithm 1, whose overhead scales linearly with the number of nodes. This can be easily mitigated with more sophisticated violation resolution protocols such as as lazy synchronization [50] and local violation resolution [34] (Section 4.2).

6 CONCLUSIONS

We presented a new approach for simultaneously approximating multiple functions over the aggregate of distributed streams (MFDS). Our approach takes advantage of the underlying unifying principles of the geometric monitoring framework: monitoring the domain of functions, and defining local convex safe zones. Unlike most single function approximations for distributed streams

(SFDS), these unifying principles allow us to easily combine existing single-function GM algorithms, while paying less than the sum of individual communication costs. Indeed, our evaluation on different sets of functions with real-world datasets demonstrates that in most cases our method results in a small communication overhead over the distributed approximation of a single function.

Clearly, different sets of approximated functions result in different common safe zones. We are exploring the inter-dependence between the functions, and its effect on CSZ.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme under grant agreements no. 619491 and no. 619435, and from the Eropean Union's Horizon 2020 Research And Innovation Programme under grant agreements no. 688380. The authors would like to thank the anonymous referees for their valuable comments and suggestions.

A APPENDIX: SAFE ZONES

This section describes the safe zones for the functions monitored in Section 5. For completeness, we include summaries of existing work (regression [19], cosine similarity [36], and inner product [36]), as well as newly-derived safe zones (condition number, percentiles, PCA-score, Euclidean distance, and norm).

A.1 Regression

Let (A, c) be as defined in Section 5.3, and similarly denote by (A_0, c_0) the estimate vector at time $t = 0$. We track $\|\beta - \beta_0\| \leq \epsilon$, where $\beta = A^{-1}c$ and $\beta_0 = A_0^{-1}c_0$. The safe zone for the regression model is [19]:

$$\left\{ (A, c) \mid \|{A_0}^{-1}\delta\| + \|{A_0}^{-1}\Delta\beta_0\| + \epsilon\|{A_0}^{-1}\Delta\| \leq \epsilon \right\} ,$$

where $\Delta = A - A_0$ and $\delta = c - c_0$.

A.2 Condition Number

For a symmetric matrix A the condition number is the ratio between maximal eigenvalue λ_1 and the minimal eigenvalue λ_m: $\kappa = \frac{\lambda_1(A)}{\lambda_m(A)}$. The task is to track the condition number with ϵ relative error: $(1-\epsilon)\kappa_0 \leq \kappa \leq (1+\epsilon)\kappa_0$, where κ_0 is the approximation and κ is the current true value of the condition number.

The upper bound is convex and can be used as is. The lower bound is neither convex nor concave. To get a convex constraint, we apply the method described in [36], replacing λ_1 and λ_m with their respective tangent planes at A_0.

Let A be as defined in Section 5.3, and similarly denote by A_0 the estimate vector at time $t = 0$. Let $H_i(A)$ be the linear approximation of $\lambda_i(A)$ around A_0: $H_i(A) \triangleq \lambda_i(A_0) + \langle e_i(A_0)e_i^t(A_0), A - A_0\rangle$. The safe zone of the condition number is:

$$\{ A \mid \lambda_1(A) \leq (1+\epsilon)\kappa_0\lambda_m(A) \wedge H_1(A) \leq (1-\epsilon)\kappa_0 H_m(A) \} .$$

A.3 PCA-Score

The kth PCA score t_k of a scatter matrix $A \in \mathbb{R}^{m\times m}$ is:

$$t_k = \frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{m}\lambda_i} .$$

The objective is to approximate t_k with an additive approximation bound ϵ: $t_{k,0} - \epsilon \leq t_k \leq t_{k,0} + \epsilon$, where t_k is the current PCA-Score and $t_{k,0}$ is the estimate vector.

Previous work has derived a safe zone for the squared PCA score [36] t_k^2 . We use similar techniques to derive safe zones for t_k directly. Using the convexity property of the sum of top-k eigenvalues, stating that for a symmetric A, the sum of top k eigenvalues $\sum_{i=1}^{k}\lambda_i$ is convex [16], and applying CB [36], we get the following safe zone for the PCA score:

$$\left\{ A \Big| \sum_{i=1}^{k} H_i(A) \geq (t_{k,0} - \epsilon)\sum_{i=1}^{m}\lambda_i(A) \wedge (t_{k,0} + \epsilon)\sum_{i=1}^{m} H_i(A) \geq \sum_{i=1}^{k}\lambda_i(A) \right\},$$

A, A_0 are as defined in Section 5.4, and $H_i(A)$ is the linear approximation of $\lambda_i(A)$ around A_0: $H_i(A) \triangleq \lambda_i(A_0)+\langle e_i(A_0)e_i^t(A_0), A-A_0\rangle$.

A.4 Distance Metrics

The task is to approximate inner product (ip), cosine similarity (csim), and Euclidean distance (euc) within a relative error of ϵ: $(1-\epsilon)f(x_0, y_0) \leq f(x, y) \leq (1+\epsilon)f(x_0, y_0)$. Let the estimate vector be $v_0 = (x_0, y_0)$ and let (x, y) be two vectors such that $x \in \mathbb{R}^m, y \in \mathbb{R}^m$. The safe zones for the functions are given below:

Safe zone for inner product. We use the safe zone from [36]:

$$\begin{aligned} \{ (x, y) \mid \|x - y\|^2 \leq\ & \|x_0 + y_0\|^2 - 4(1-\epsilon)\langle x_0, y_0\rangle + \\ & 2\langle (x_0 + y_0, x_0 + y_0), (x - x_0, y - y_0)\rangle , \\ \|x + y\|^2 \leq\ & \|x_0 - y_0\|^2 + 4(1+\epsilon)\langle x_0, y_0\rangle + \\ & 2\langle (x_0 - y_0, y_0 - x_0), (x - x_0, y - y_0)\rangle \} . \end{aligned}$$

Safe zone for cosine similarity. Let $f(x, y) = \frac{\langle x, y\rangle}{\|x\|\|y\|}$ be the cosine similarity function, and define two convex axillary functions g, h:

$$\begin{aligned} g(x, y) &= \|x - y\|^2 + (1-\epsilon)f(x_0, y0)\left(4\|x\|\|y\| + 2(\|x\|^2 + \|y\|^2)\right) \\ h(x, y) &= \|x + y\|^2 + 2(1+\epsilon)f(x_0, y0)(\|x\|^2 + \|y\|^2) \quad . \end{aligned}$$

The safe zone for cosine similarity is given by [36]:

$$\begin{aligned} \{ (x, y) \mid & h(x, y) \geq g(x_0, y_0) + \langle \nabla g(x_0, y_0), (x - x_0, y - y_0)\rangle, \\ & g(x, y) \geq h(x_0, y_0) + \langle \nabla h(x_0, y_0), (x - x_0, y - y_0)\rangle \} , \end{aligned}$$

where $\nabla h(x_0, y_0)$ and $\nabla g(x_0, y_0)$ are the gradients of h and g at (x_0, y_0), respectively.

Safe zone for Euclidean distance. Let $f(x, y) = \|x - y\|^2$ be the euclidean distance function. The upper bound $f(x, y) \leq (1+\epsilon)f(x_0, y_0)$ is convex. The lower bound $f(x, y) \geq (1-\epsilon)f(x_0, y_0)$ is not convex, so we apply the method in [36] to derive a convex lower bound. Using both convex bounds the safe zone for the euclidean distance is:

$$\begin{aligned} \{ (x, y) \mid & ((1+\epsilon)\|x_0 - y_0\|)^2 \geq \|x - y\|^2, \\ & ((1-\epsilon)f(x_0, y_0))^2 \leq \|x_0 - y_0\|^2 + \\ & \qquad 2\langle (x_0 - y_0, y_0 - x_0), (x - x_0, y - y_0)\rangle \} . \end{aligned}$$

A.5 Percentiles

For tracking percentiles, we derive convex lower and upper bounds, and define the safe as the set of all vectors $\{x\}$ that satisfy both. This generalizes [37], where a safe zone for the median ($p = 50$) was derived using the CD method.

We start with describing the lower bound. Given a percentile $0 \leq p \leq 100$, the global estimate $v_0 \in \mathbb{R}^n$, the local vector $v \in \mathbb{R}^n$, and the lower bound $T = (1-\epsilon)f_p(v_0)$, define: $\hat{x_0} = (v_{01} - T, v_{02} - T, \ldots, v_{0n} - T)$ and $\hat{v} = (v_1 - T, v_2 - T, .., v_n - T)$. Let L be the number of positive $\hat{r}_i$, and let s be the sorted vector of the products $\hat{v_{0i}}\hat{v_i}$: $s = \text{Sort}\left(\{\hat{v_{0i}}\hat{v_i} | 1 \leq i \leq N\}\right)$, then the lower bound constraint is: $\sum_{i=1}^{N-L+1} s_i \geq 0$. To derive the upper bound, invert the signs of v_0 and v, set $T = -(1+\epsilon)f_p(v_0)$, and continue as before.

A.6 Norm

Let $f(v) = \|v\|$ be the norm function, and let v_0 be the estimate vector. We aim to track the norm within a relative approximation bound ϵ: $(1-\epsilon)\|v_0\| \leq \|v\| \leq (1+\epsilon)\|v_0\|$. The upper bound $\|v\| \leq (1+\epsilon)\|v_0\|$ is convex. We apply convex decomposition [37] to derive a convex lower bound: $\langle v_0 - (1-\epsilon)v_0\,,\, v - (1-\epsilon)v_0 \rangle \geq 0$. The safe zone for the norm function is a combination of both bounds: $\{\, v \mid \|v\| \leq (1+\epsilon)\|v_0\|,\, \langle v_0 - (1-\epsilon)v_0\,,\, v - (1-\epsilon)v_0 \rangle \geq 0 \,\}$.

REFERENCES

[1] Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. 1999. Tracking Join and Self-Join Sizes in Limited Storage. In *PODS '99*.
[2] Chrisil Arackaparambil, Joshua Brody, and Amit Chakrabarti. 2009. Functional monitoring without monotonicity. In *ICALP '09*.
[3] Arvind Arasu and Jennifer Widom. 2004. Resource sharing in continuous sliding-window aggregates. In *VLDB '04*.
[4] B. Babcock and C. Olston. 2003. Distributed top-k monitoring. In *SIGMOD '03*.
[5] Amir Bar-Or, Daniel Keren, Assaf Schuster, and Ran Wolff. 2005. Hierarchical decision tree induction in distributed genomic databases. *TKDE* (2005).
[6] Steven Bird, Ewan Klein, and Edward Loper. 2009. Natural Language Processing with Python. O'Reilly Media.
[7] Peter Bodik, Wei Hong, Carlos Guestrin, Sam Madden, Mark Paskin, and Romain Thibaux. 2004. Intel Lab Data. (2004). http://db.csail.mit.edu/labdata/labdata.html
[8] Andries E Brouwer and Willem H Haemers. 2011. *Spectra of graphs*. Springer Science & Business Media.
[9] Sabbas Burdakis and Antonios Deligiannakis. 2012. Detecting Outliers in Sensor Networks Using the Geometric Approach. In *ICDE '12*.
[10] Richard Cangelosi and Alain Goriely. 2007. Component retention in principal component analysis with application to cDNA microarray data. *Biology direct* (2007).
[11] Graham Cormode. 2011. Sketch techniques for approximate query processing. *Synposes for Approximate Query Processing: Samples, Histograms, Wavelets and Sketches, Foundations and Trends in Databases. NOW publishers* (2011).
[12] Graham Cormode. 2013. The continuous distributed monitoring model. *SIGMOD Record* (2013).
[13] Graham Cormode and Minos Garofalakis. 2008. Approximate continuous querying over distributed streams. *TODS* (2008).
[14] Graham Cormode, S Muthukrishnan, and Ke Yi. 2011. Algorithms for distributed functional monitoring. *TALG* (2011).
[15] Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. 2004. Sketch-based multi-query processing over data streams. In *EDBT '04*.
[16] Ky Fan. 1949. On a Theorem of Weyl Concerning Eigenvalues of Linear Transformations I. In *PANS '49*.
[17] Arik Friedman, Izchak Sharfman, Daniel Keren, and Assaf Schuster. 2014. Privacy-Preserving Distributed Stream Monitoring.. In *NDSS '14*.
[18] Moshe Gabel, Daniel Keren, and Assaf Schuster. 2013. Communication-efficient Outlier Detection for Scale-out Systems.. In *BD3 @ VLDB*.
[19] Moshe Gabel, Daniel Keren, and Assaf Schuster. 2015. Monitoring least squares models of distributed streams. In *KDD '15*.
[20] Moshe Gabel, Daniel Keren, and Assaf Schuster. 2017. Anarchists, Unite: Practical Entropy Approximation for Distributed Streams. In *KDD '17*.
[21] Moshe Gabel, Assaf Schuster, and Daniel Keren. 2014. Communication-efficient distributed variance monitoring and outlier detection for multivariate time series. In *IPDPS '14*.
[22] Minos Garofalakis and Vasilis Samoladas. 2017. Distributed Query Monitoring through Convex Analysis: Towards Composable Safe Zones. In *ICDT '17*.
[23] Minos N. Garofalakis, Daniel Keren, and Vasilis Samoladas. 2013. Sketch-based Geometric Monitoring of Distributed Stream Queries. *PVLDB* (2013).
[24] Nikos Giatrakos, Antonios Deligiannakis, and Minos Garofalakis. 2016. Scalable approximate query tracking over highly distributed data streams. In *SIGMOD'16*.
[25] Nikos Giatrakos, Antonios Deligiannakis, Minos Garofalakis, Izchak Sharfman, and Assaf Schuster. 2012. Prediction-based geometric monitoring over distributed data streams. In *SIGMOD '12*.
[26] Nikos Giatrakos, Antonios Deligiannakis, Minos Garofalakis, Izchak Sharfman, and Assaf Schuster. 2014. Distributed geometric query monitoring using prediction models. *TODS* (2014).
[27] Shenoda Guirguis, Mohamed A Sharaf, Panos K Chrysanthis, and Alexandros Labrinidis. 2011. Optimized processing of multiple aggregate continuous queries. In *CIKM '11*.
[28] Ling Huang, XuanLong Nguyen, Minos N. Garofalakis, Joseph M. Hellerstein, Michael I. Jordan, Anthony D. Joseph, and Nina Taft. 2007. Communication-Efficient Online Detection of Network-Wide Anomalies. In *INFOCOM '07*.
[29] Ryan Huebsch, Minos Garofalakis, Joseph M Hellerstein, and Ion Stoica. 2007. Sharing aggregate computation for distributed queries. In *SIGMOD '07*.
[30] Michael Kamp, Mario Boley, Daniel Keren, Assaf Schuster, and Izchak Sharfman. 2014. Communication-efficient distributed online prediction by dynamic model synchronization. In *ECML '14*.
[31] Srinivas R. Kashyap, Jeyashankher Ramamirtham, Rajeev Rastogi, and Pushpraj Shukla. 2008. Efficient Constraint Monitoring Using Adaptive Thresholds. In *ICDE '08*.
[32] KDD. 1999. KDD99 cup dataset. (1999). https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[33] Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. 2006. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD '06*.
[34] Daniel Keren, Guy Sagy, Amir Abboud, David Ben-David, Assaf Schuster, Izchak Sharfman, and Antonios Deligiannakis. 2014. Geometric monitoring of heterogeneous streams. *TKDE* (2014).
[35] Daniel Keren, Izchak Sharfman, Assaf Schuster, and Avishay Livne. 2012. Shape Sensitive Geometric Monitoring. *TKDE* (2012).
[36] Arnon Lazerson, Daniel Keren, and Assaf Schuster. 2016. Lightweight Monitoring of Distributed Streams. In *KDD '16*.
[37] Arnon Lazerson, Izchak Sharfman, Daniel Keren, Assaf Schuster, Minos N. Garofalakis, and Vasilis Samoladas. 2015. Monitoring Distributed Streams using Convex Decompositions. *PVLDB* (2015).
[38] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. 2004. Rcv1: A new benchmark collection for text categorization research. *JMLR* (2004).
[39] Rui Li, Shengjie Wang, Hongbo Deng, Rui Wang, and Kevin Chen-Chuan Chang. 2012. Towards social user profiling: unified and discriminative influence model for inferring home locations. In *KDD '12*.
[40] Wen Liu, Yan-Ming Shen, and Peng Wang. 2016. An Efficient Approach of Processing Multiple Continuous Queries. *JCSIT* (2016).
[41] Charlotte H Mason and William D Perreault Jr. 1991. Collinearity, power, and interpretation of multiple regression analysis. *JMR* (1991).
[42] Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. 2005. KLEE: a framework for distributed top-k query algorithms. In *VLDB '05*.
[43] Shanmugavelayutham Muthukrishnan and others. 2005. Data streams: Algorithms and applications. *FnT-TCS* (2005).
[44] Noam Palatin, Arie Leizarowitz, Assaf Schuster, and Ran Wolff. 2008. Mining for misconfigured machines in grid systems. *Data Mining Techniques in Grid Computing Environments* (2008).
[45] Themistoklis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. 2003. Distributed deviation detection in sensor networks. *SIGMOD Record* (2003).
[46] Guy Sagy, Daniel Keren, Izchak Sharfman, and Assaf Schuster. 2010. Distributed threshold querying of general functions by a difference of monotonic representation. *PVLDB* (2010).
[47] Aušra Saudargienė. 1999. Structurization of the covariance matrix by process type and block-diagonal models in the classifier design. *Informatica* (1999).
[48] Shetal Shah and Krithi Ramamritham. 2008. Handling Non-linear Polynomial Queries over Dynamic Data. In *ICDE '08*.
[49] Izchak Sharfman, Assaf Schuster, and Daniel Keren. 2007. Aggregate threshold queries in sensor networks. In *IPDPS '07*.
[50] Izchak Sharfman, Assaf Schuster, and Daniel Keren. 2007. A geometric approach to monitoring threshold functions over distributed data streams. *TODS* (2007).
[51] Izchak Sharfman, Assaf Schuster, and Daniel Keren. 2008. Shape sensitive geometric monitoring. In *PODS '08*.
[52] Daniel A Spielman. 2007. Spectral graph theory and its applications. In *FOCS'07*.
[53] Ke Yi and Qin Zhang. 2009. Optimal tracking of distributed heavy hitters and quantiles. In *PODS '09*.