Lightweight Monitoring of Distributed Streams

ARNON LAZERSON, Technion – Israel Institute of Technology, Israel
DANIEL KEREN, University of Haifa, Israel
ASSAF SCHUSTER, Technion – Israel Institute of Technology, Israel

As data becomes dynamic, large, and distributed, there is increasing demand for what have become known as *distributed stream algorithms*. Since continuously collecting the data to a central server and processing it there is infeasible, a common approach is to define *local* conditions at the distributed nodes, such that—as long as they are maintained—some desirable *global* condition holds.

Previous methods derived local conditions focusing on communication efficiency. While proving very useful for reducing the communication volume, these local conditions often suffer from heavy computational burden at the nodes. The computational complexity of the local conditions affects both the runtime and the energy consumption. These are especially critical for resource-limited devices like smartphones and sensor nodes. Such devices are becoming more ubiquitous due to the recent trend toward smart cities and the Internet of Things. To accommodate for high data rates and limited resources of these devices, it is crucial that the local conditions be quickly and efficiently evaluated.

Here we propose a novel approach, designated CB (for Convex/Concave Bounds). CB defines local conditions using suitably chosen convex and concave functions. Lightweight and simple, these local conditions can be rapidly checked on the fly. CB's superiority over the state-of-the-art is demonstrated in its reduced runtime and power consumption, by up to six orders of magnitude in some cases. As an added bonus, CB also reduced communication overhead in all the tested application scenarios.

CCS Concepts: • **Computing methodologies → Distributed algorithms**; • **Information systems →** *Data stream mining*; *Parallel and distributed DBMSs*;

Additional Key Words and Phrases: Disributed stream mining, continuous distributed monitoring

ACM Reference format:
Arnon Lazerson, Daniel Keren and Assaf Schuster. 2018. Lightweight Monitoring of Distributed Streams. *ACM Trans. Database Syst.* 43, 2, Article 9 (July 2018), 37 pages.
https://doi.org/10.1145/3226113

1 INTRODUCTION

Continuous, real-time processing of vast amounts of rapidly changing data lies at the heart of many modern and emerging applications. Examples include health monitoring over the IoT [31],

The research leading to these results has received funding from the [European Union's] Seventh Framework Programme [FP7-ICT-2013-11] under Grant Agreement No. 619491 and No. 619435 and from the European Commission Horizon 2020-the Framework Programme for Research and Innovation (2014-2020) under Grant Agreement No. 688380.

Authors' addresses: A. Lazerson, Technion – Israel Institute of Technology, Haifa 32000, Israel; email: lazerson@cs.technion.ac.il; D. Keren, University of Haifa, Haifa 31905, Israel; email: dkeren@cs.haifa.ac.il; A. Schuster, Technion – Israel Institute of Technology, Haifa 32000, Israel; email: assaf@cs.technion.ac.il.

https://doi.org/10.1145/3226113

smart city infrastructures [33], financial data analysis [67, 68], social media stream mining for recommendation systems [5], and other application scenarios.

The distributed nature of the data streams, the massive amount of information they carry, as well as the real-time processing requirements introduce some fundamental challenges. The main challenge is reducing communication volume [13, 14, 38]. Continuously collecting the data to a central location is infeasible in large scale applications, as the excess communication required interferes with the normal operation of the data network [36]. Furthermore, in the case of battery-operated devices such as WSN sensor nodes, central data accumulation depletes the power supply of individual devices, reducing the network lifetime [46]. Another key challenge is processing high-speed data streams given the runtime limitations of the remote sites [12]. This is especially true for resource-limited devices such as sensor nodes, where the CPU is weak and memory and storage are scarce.

Continuous data stream systems require different processing paradigms than traditional systems where persistent *datasets* are stored [4]. Instead of traditional one-shot queries, *continuous queries* [4, 59] are issued by the user. The system continuously evaluates these queries, providing the user with updated results.

An important class of distributed continuous queries are the threshold monitoring queries. A threshold on the value of a function over the union of the distributed stream is defined, and the system must issue an alert when this threshold is crossed [29, 32, 36, 59, 60]. Examples include detecting when the sum of a distributed set of variables exceeds a predetermined threshold [15], or checking whether the value of the information gain function globally exceeds a given threshold to detect spam in distributed mail systems [59].

The problem of effectively evaluating threshold monitoring queries over continuous distributed streams is known as the *distributed monitoring problem* (also referred to as the *functional monitoring problem*, [9, 53, 63]; see also the survey in Reference [11]). It can be broadly defined as follows:

Definition 1.1. Given is a distributed system, with nodes $N_1 \ldots N_k$, with N_i holding a dynamic data vector $v_i(t)$ (t will be suppressed hereafter to reduce equation clutter). Also given is a function f, which depends on all the v_i's, and a threshold T. The goal is to define *local* conditions at the nodes, such that:

— *Correctness*: As long as all local conditions hold, the global condition $f(v_1 \ldots v_k) \leq T$ is also guaranteed to hold.
— *Communication efficiency*: The local conditions are "lenient," i.e., the number of times they are violated is minimal.
— *Computational efficiency*: The complexity of checking the local conditions is minimal.

As a motivating real-life example that applies the Pearson Correlation Coefficient (PCC) function (treated in this article), consider a distributed sensor network used to monitor air quality [49]. Often, not only is the information on the individual pollutants important but also the *correlations* between them. For example, if an *event i* is defined as pollutant i crossing a certain threshold, one may wish to know whether there exists a correlation between events i, j for two different pollutants. A commonly used measure, the PCC quantifies such a correlation by the value $\frac{z-xy}{\sqrt{x-x^2}\sqrt{y-y^2}}$, where x, y, z are, respectively, the probabilities of event i, event j, and both events simultaneously. For a distributed system, the global probabilities are averaged over the nodes. It is easy, however, to see that the PCC value of the *global* probabilities can be above a given threshold T, while the *local* value at some of the nodes is below T, and vice versa (for example, in a system with two nodes and local values $x_1 = 0.8, y_1 = 0.2, z_1 = 0.17$ and $x_2 = 0.2, y_2 = 0.7, z_2 = 0.15$, the local PCC

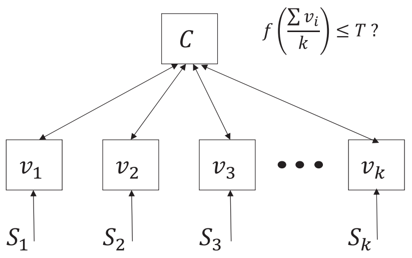

Fig. 1. Distributed monitoring model: Distributed streams S_i continuously update the local vectors v_i. The coordinator C must issue an alert when the global condition $f(\frac{v_1+..+v_k}{k}) \leq T$ is breached.

values are 0.062 and 0.054, and the global value is −0.26). This is because, for arbitrary functions, there is generally no correlation between the *average of the values* and the *value at the average*.

For general functions defined over a distributed system, it is typically impossible to determine the position of their global values vis-à-vis T, when given just the local values. The distributed monitoring problem is to impose local conditions guaranteeing that the global value did not cross T. This problem is known to be rather difficult (NP-complete even in very simple scenarios; see Reference [37]). Nonetheless, considerable progress has been made for real-life problems (Section 2).

Sharfman et al. [59] introduced a distributed model where the monitored query can be expressed as the application of an arbitrary function to the aggregated vector $\frac{v_1+..+v_k}{k}$, i.e., $f = f(\frac{v_1+..+v_k}{k})$ (see Figure 1). This model turns out to be rich enough to be applicable to a wide range of problems (see Section 2.1). Further, it can be extended by augmenting the local vectors by various functions of the coordinates, allowing it to handle a rather wide class of functions [10, 21].

While considerable progress was made in reducing the communication cost in this distributed model, computational cost reduction received little attention. Our goal in this work is to improve the computational cost while maintaining communication costs similar to those attained by the state-of-the-art. Our evaluations show that our method not only improved runtime by up to six orders of magnitude, but it also achieved better communication costs than previous work.

To reduce the computational complexity, we propose here a simple and direct method to solve the distributed monitoring problem. It relies on the simple observation that, if f is a *convex* function, then, if $f(v_i) \leq T$ holds at every node, it also holds that $f(\frac{v_1+..+v_k}{k}) \leq T$. Thus, monitoring a convex function (from above) is trivial – just monitor its value at every node.

To handle a general f, we propose to search for a convex function c such that $c(u) \geq f(u)$ for all vectors u, and monitor the condition $c \leq T$. This yields a simple monitoring condition, whose correctness implies the correctness of the desired condition $f \leq T$. Naturally, the following conditions should hold:

— c should be easy to derive and calculate.
— To avoid a high ratio of "false alarms", c should tightly bound f.[1]

We refer to the proposed method as *convex bound* (CB). Clearly, $f \geq T$ can be similarly monitored by finding a *concave* lower bound.

The recent trend toward smart cities and the Internet of Things (IoT) depends to a large extent on the deployment of ubiquitous nodes built of sensors and reduced computation systems. Such nodes, which we refer to as "thin," are extremely resource-constrained. They have reduced CPU capabilities, small memory, and limited local storage. In addition, they often have to communicate

[1] As will be shown later, the choice of c also depends on the initial value of the average vector $\frac{v_1+..+v_k}{k}$.

over wireless media, and may be powered by batteries—which means they will be highly restricted by energy considerations. In smart environments, however, resource-limited nodes are expected to support important collaborative, continuous monitoring tasks. We believe that the CB method introduced here, being both lightweight and communication efficient, will enable this goal.

In this article, we make the following contributions: (1) we introduce the CB method and apply it to monitor four popular functions—the PCC, inner product, cosine similarity, and PCA-Score; (2) we experimentally validate CB against state-of-the-art methods, demonstrating CB's superiority in both computation and communication costs; and (3) we implement CB on resource-limited devices, which results in significant savings in battery lifetime.

2 RELATED WORK

Much early work on monitoring distributed streams dealt with the simpler cases of linear functions [35, 36], as well as monotonic functions [47], and counting distinct elements [6]. Non-monotonic functions were addressed in Reference [55] by representing them as a difference of monotonic functions. Distributed sensor networks were studied in References [50, 51, 58]. Other work included top-k monitoring [3], distributed monitoring of the value of a single-variable polynomial [56], and perturbative analysis of eigenvalues, which was applied to determine local conditions on traffic volume data at the nodes of a distributed system to monitor system health [29]. In Reference [62], the monitoring problem was studied in a probabilistic setting, and, in addition to the function's score, a probability threshold was applied; see also Reference [44]. Monitoring entropy was studied in Reference [2]. Ratio queries were handled in Reference [26]. In Reference [64], the norm of the average vector was monitored.

While some problems in monitoring over distributed systems were treated in the past, we are not aware of any general method (capable of handling arbitrary non-linear, non-monotonic, non-convex functions) except for *geometric monitoring* (GM) and its derivatives, which are surveyed next.

2.1 Previous Work on Geometric Monitoring

In References [57, 59], a general approach, GM, was proposed for tracking the value of a *general* function over distributed streams. GM rests on a geometric result, the so-called *bounding lemma* (details follow in this subsection), which makes it possible to "break up" a global threshold query into conditions that can be checked locally at each site. Follow-up work [38] proposed various extensions to the basic method.

GM achieved impressive results in reducing communication overhead for a nice range of central problems, including efficient outlier detection in sensor networks [10]; sketch-based monitoring of norm, range-aggregate, and join-aggregate queries over distributed streams [22]; efficiently computing and tracking skylines in a distributed setting [52]; tracking least squares regression models [19]; reducing channel state information in distributed networks [30]; and approximating entropy of distributed streams [20]. Other recent work included an extension to predictive data modeling [23, 24], treatment of heterogeneous streams [37], and a privacy-preserving variant [18].

We use GM as a baseline for our comparison since it achieved state-of-the-art results in reducing communication overhead where applied. Unfortunately, its application is typically hampered by high computational overhead at the nodes. It is this problem that the proposed CB approach aims to alleviate. GM is briefly described next. Proofs and further details can be found in Reference [38] (which is the version that was implemented). Since both GM and CB heavily rely on the notion of convexity, we start with some relevant definitions (see Reference [8]).

A set is convex iff it satisfies the following property: if two points are inside it, so is the line segment between them. A function is convex iff the region above its graph is convex. Formally:

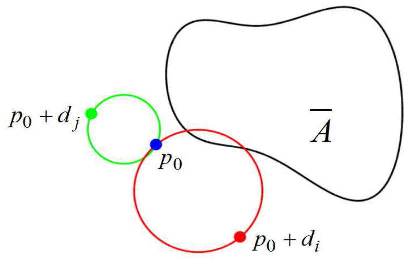

Fig. 2. Applying local conditions in GM. The drift vector d_i causes a violation, since the sphere it defines with p_0 intersects the inadmissible region $\bar{A}$; however, d_j does not cause a violation.

(1) A *convex combination* of points u_i in Euclidean space is an expression of the form $\sum_i \lambda_i u_i$, where the λ_i's are positive scalars whose sum equals 1.
(2) A set will be called *convex* if it contains all the convex combinations of all its finite subsets.
(3) The *convex hull* of a set B is the smallest (w.r.t. inclusion) convex set that contains B.
(4) A real-valued function f will be called convex iff, for every convex combination $\sum_i \lambda_i u_i$, the following holds: $f(\sum_i \lambda_i u_i) \leq \sum_i \lambda_i f(u_i)$.
(5) For every convex function and every threshold T, the set $\{u|f(u) \leq T\}$ is convex.
(6) f will be called *concave* iff $(-f)$ is convex.
(7) If f is convex (resp. concave), it lies above (resp. below) all its tangent planes.

A Brief View of GM. Recall that the distributed monitoring problem considers whether $(f(\frac{v_1+..+v_k}{k}) \leq T$, where $\{v_i\}$ denote the local dynamic data vectors at the nodes. GM rests on the following geometric interpretation of this question: define the *admissible region*, A, by $A \triangleq \{u|f(u) \leq T\}$. Then, the question is whether the condition $\frac{v_1+..+v_k}{k} \in A$ holds. The first step in answering this question is the following:

LEMMA 2.1. *[59] Let $v_i(0)$ denote the initial value of the data vector at the i-th node, and let the so-called* reference point, *p_0, be equal to the average of these initial values: $p_0 = \frac{v_1(0)+..+v_k(0)}{k}$. We assume that, during the initial synchronization, a coordinator node broadcasts p_0 to all nodes. Denote the change in the data vector at the i-th node, i.e., $v_i - v_i(0)$, by d_i (it will be referred to as the i-th* drift vector*). Then, the following holds:*

$$v = \frac{v_1 + .. + v_k}{k} = \frac{(p_0 + d_1) + .. + (p_0 + d_k)}{k}.$$

Now, the i-th node can independently compute $p_0 + d_i$, and since the global vector v is equal to the average of $p_0 + d_i$, $i = 1..k$, it obviously lies in their convex hull. It is therefore desirable to impose *local* conditions on $p_0 + d_i$, which will guarantee that $v \in A$. This is achieved by the *bounding lemma*:

THEOREM 2.2. *[59] Let B_i denote the (solid) sphere with center $p_0 + d_i/2$ and radius $\|d_i\|/2$. Then the union $\bigcup_{i=1}^{k} B_i$ contains the convex hull of the vectors $p_0, p_0 + d_1 ... p_0 + d_k$; hence it contains v.*

As a result of the bounding lemma, the local condition used in GM is the following: node i remains silent as long as its sphere B_i is contained in A (Figure 2). If this condition is violated – that is, B_i intersects the *inadmissible region*[2] $\bar{A}$, the system enters a *violation recovery* phase.

[2]The inadmissible region marked by $\bar{A}$ is the complement of the admissible region, A.

2.1.1 Violation Recovery. GM guarantees that any global violation will be indicated by a local violation. Note, however, that a local violation does not necessarily indicate a global violation, since it is possible that $f(p_0 + d_i) > T$ but $f(v) \leq T$. Such false violations are unavoidable in a distributed system, where a node can access only its local data (this is true even if the admissible region itself is convex). From the node's point of view, there is no difference between a local and a global violation, therefore it must notify the coordinator whenever a local violation (i.e., $f(p_0 + d_i) > T$) occurs. It is the responsibility of the coordinator to collect more data from the nodes to decide whether this local violation indicates a global violation or not.

Once the coordinator has been notified of a local violation, it can apply one of several strategies to resolve it. The simplest violation resolution strategy is known as "eager synchronization" [22, 59]. The coordinator collects the local data (v_i) from all nodes, calculates $v = \frac{1}{k} \sum_1^k v_i$ and checks whether global violation indeed occurred ($f(v) > T$). If there is a global violation, the coordinator issues an alert, otherwise the coordinator distributes a new reference point $p_0 = v$ to all nodes, and the monitoring process continues.

"Lazy synchronization" [22, 59] is a more complicated violation resolution strategy that scales better with the number of nodes. The coordinator attempts to resolve the violation by searching for a subset of nodes (which contains the violating node) whose local vectors "balance" each other (i.e., the value of the bounding function evaluated at their average is below the threshold). If no such subset exists, the coordinator issues an alert if there is a true global violation, and sends a new reference point if there isn't (same as "eager synchronization").

In this article, we applied the "lazy" recovery scheme, in which the coordinator gradually gathers local vectors until it manages to balance the violating ones.

2.2 Computational Complexity of GM

To apply GM, it must be repeatedly checked whether a certain sphere intersects with $\bar{A}$—that is, whether its radius is smaller than the distance from its center to A's boundary, which is defined by the *threshold surface* $\{u | f(u) = T\}$. Finding the distance from a point to a threshold surface of a general function is notoriously difficult, and a closed-form solution very rarely exists. Worse, there exist no algorithms that guarantee that the distance (and the closest point) will be found for every function. Even for the case of a polynomial f, the solution may require an inordinate amount of time; closed-form solutions are often impossible to derive, and iterative schemes are slow and not guaranteed to converge (see a recent survey in Reference [54]). For example, using state-of-the-art solvers to find the closest point to the surface defined by the cosine similarity function (Section 4.3), required roughly 3 minutes, and a closed-form solution does not exist even for the lowest-dimensional case (cosine similarity between two-dimensional vectors). Known upper bounds on the complexity of the closest point problem are extremely high. Solving via Lagrange multipliers yields a system of equations that may be very difficult to solve even for the relatively simple case in which the surface is described by an algebraic equation; upper bounds on the complexity are doubly exponential in the number of variables.[3]

2.3 Convex Decomposition

In Reference [42], GM was extended by the *convex decomposition* (CD) approach, which works by decomposing $\bar{A}$ into convex subsets, and which is applied to monitor sketches over distributed streams. However, the resulting algorithm has to be specifically tailored to each monitored function, and it suffers from the need to solve the same type of problem as GM (finding the closest

[3]See survey at http://tinyurl.com/lr4zhrk.

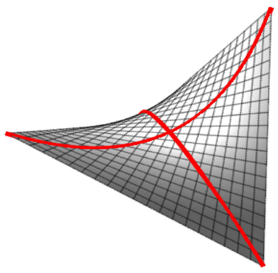

Fig. 3. Every point on the boundary of the admissible region defined by $\{(x, y, z) | z < xy\}$ is a saddle point—it has at least one negative and at least one positive curvature.

point on a surface). For the inner product function, the solution was quite complicated, and we could not apply CD to the PCC or to cosine similarity, which are easily treated by CB.

CD works by decomposing the inadmissible region $\bar{A}$ into a union of convex subsets and then separating each of them from the reference point p_0 by a suitable half-space. The intersection of all half-spaces defines a convex subset of the admissible region, which is used for monitoring.

Thus, to apply CD, it is necessary to find a CD of $\bar{A}$. Obviously, if either $\bar{A}$ or A is convex, the solution is trivial. However, from the following lemma it follows that it is typically impossible to find such a *finite* decomposition:

Lemma 2.3. *If every point on the boundary of the admissible region is a* saddle point*, that is, has at least one negative and at least one positive curvature, then neither the admissible region nor its complement can be partitioned into a finite union of convex sets.*

A very simple example of such a set is the collection of all points $\{(x, y, z\}$ in three-dimensional space that satisfy $z \leq xy$ (see Figure 3). Obviously, it is trivial to construct such surfaces in higher dimensions, for example $x_n = \pm x_1^2 \pm x_2^2 \pm \cdots \pm x_{n-1}^2$, with at least one of the x_i^2 having a +1 coefficient and at least one having a −1 coefficient.

The proof requires a considerable foray into the realm of differential geometry [16], and cannot be provided here; alas the intuition behind it is relatively simple. At a saddle point, there is at least one curve on the surface (with positive curvature) that "bends inward," and at least one curve (with negative curvature) that "bends outward." Thus, the tangent plane at the saddle point will always intersect *both* the admissible region and its complement, and it is therefore possible to partition it only to a *continuum* of convex sets, since lower dimensional "slices" of the original surface may be saddle point free (see Figure 21 in Reference [42]).

Therefore, save for the degenerate case in which f is convex, concave, or piecewise linear, it will be exceedingly difficult to apply the CD method, as an infinite number of constraints is required to define the convex subset of A. Thus, CD can be applied successfully to handle functions such as median, percentiles, and min/max, but not general functions. For example, no finite decomposition exists for any of the functions treated in this article.

3 THE CB METHOD

Runtimes such as those often incurred by GM are unacceptable for many distributed streaming systems, especially those over resource-limited nodes. The CB method presented in this article is designed to alleviate this problem. In this section, we present the main idea behind the CB method and prove some results concerning its application; in Section 4, we apply CB to four popular functions; and in Sections 5–8, we evaluate CB and demonstrate its advantage over GM.

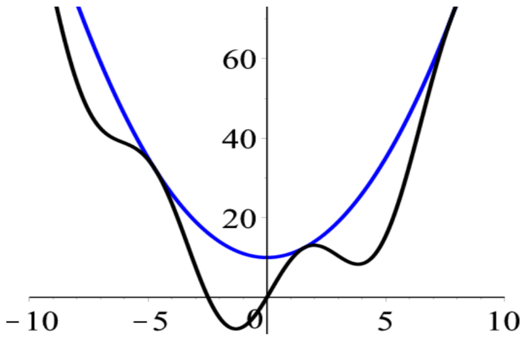

Fig. 4. $c(x) = x^2 + 10$ (blue curve) is a convex bound for $f(x) = x^2 + 10\sin(x)$ (dark curve). Given a threshold T, $c(x) \leq T$ implies $f(x) \leq T$.

A conference version of this article appeared in Reference [41], where the initial CB framework was presented. This version includes important proofs missing from the conference version; a review of the applicability and limitations of the framework; a deeper discussion concerning communication reduction; and a more extensive evaluation, including comparison to the new common safe zone (CSZ) method [40], and power consumption results on resource-limited devices.

3.1 Basics of CB

As noted in the Introduction, it is easy to define local conditions for the monitoring problem (Definition 1.1) when f is convex: every node i must only check the condition $f(p_0 + d_i) \leq T$ (correctness follows immediately from Lemma 2.1 and property 4 in Section 2.1). We propose to extend this simple observation to monitor arbitrary functions, using an approach that works directly in the realm of functions as opposed to seeking a geometric solution. The proposed solution works by "relaxing" f to a convex function c that bounds f from above and monitoring the condition $c \leq T$. This condition implies $f \leq T$ and is also easy to monitor. We shall refer to c as a *convex bound* for f. Figure 4 schematically demonstrates the idea behind CB.

The next theorem states that every solution that GM provides is also realizable as a solution provided by CB.

Theorem 3.1. *For every monitoring problem, there is a solution obtained with CB that is exactly identical to the solution obtained with GM—that is, it imposes exactly the same local conditions.*

Proof. Let $f \leq T$ be any monitoring problem. As proven in previous work (i.e., Reference [38]), the GM monitoring algorithm defines some convex subset of the admissible region A, call it C, and then checks whether $p_0 + d_i \in C$. In other words, the sphere containment condition (Section 2.1, Figure 2) describes a convex subset of the admissible region.

We seek to prove that any such C can be realized in the CB framework, i.e., there exists a convex bound $c > f$ (where $c > f$ means that for every u, $c(u) \geq f(u)$) such that, for any point u, $u \in C$ iff $c(u) \leq T$.

Such a function c can be defined using the *distance transform* for the set C. We recall that this function—denoted d_C – is defined as follows: if $u \in C$ then $d_C(u) = 0$, and if $u \notin C$, then $d_C(u)$ is the distance from u to C, i.e., the distance from u to the closest point in C. Then, we simply define $c(u) = T + d_C(u)$. Clearly, $c \leq T$ exactly on T, and $c > f$ on C. The function c looks like a bowl with a bottom in the shape of C; see Figure 5.

To conclude the proof, we only need the following lemma [48]:

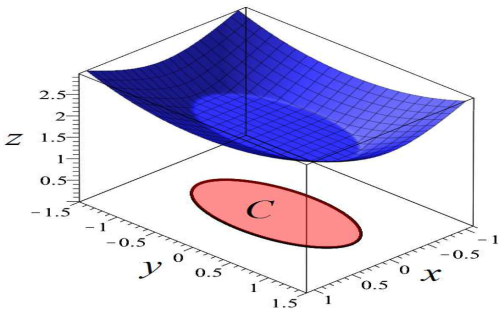

Fig. 5. A schematic description for the proof of Theorem 3.1. The set C is the ellipse, $T = 2$, and the function c is the blue surface; it is convex, and further, it is $\geq T$ exactly inside C. Such a function can be built for every convex C.

Lemma 3.2. *The distance transform of a convex set is convex.*

3.2 Choosing Convex Bounds

There are, of course, an infinite number of convex bounds for f, and the question is which of them to choose. To this end, we first propose the following definition.

Definition 3.3. Let f be the monitored function. A *tight family* of convex bounds for f, denoted $\mathcal{CB}(f)$, is a set of convex functions satisfying the following requirements:

- $g \in \mathcal{CB}(f)$ implies that g is convex and, for every u, $g(u) \geq f(u)$ (the last condition will be denoted $g \succ f$).
- Let c be any convex function such that $c \succ f$. Then there exists $g \in \mathcal{CB}(f)$ such that $g \prec c$.
- If $g_1, g_2 \in \mathcal{CB}$, then neither $g_1 \prec g_2$ nor $g_2 \prec g_1$.

Clearly, if g_1, g_2 are both convex bounds for f, and $g_1 \prec g_2$, it is better to use g_1 when monitoring f (since the condition $g_2(v) \leq T$ is weaker than $g_1(v) \leq T$). Therefore we have the following:

Observation. When applying CB to monitor f, the convex bound should belong to some family of tight bounds of f.

In the following case, it is possible to define $\mathcal{CB}(f)$:

Lemma 3.4. *Let f be a concave function. Then the family of all tangent planes to f defines a family of tight bounds.*[4]

Proof. Every tangent plane is linear, hence convex. Further, it is known that a concave function lies under any of its tangent planes. Now, let g be convex and $g \succ f$. Denote by $U(g)$ the set of all points above $g's$ graph, and by $B(f)$ all points below $f's$ graph. Then both $U(g), B(f)$ are convex, and the minimal distance between them is therefore obtained at points on their boundaries. The tangent plane at the point on f's boundary is the desired element of $\mathcal{CB}(f)$. The idea of the proof is outlined in Figure 6. □

3.3 The Convexity Gap and Dependence on the Reference Point

Replacing the monitored condition $f \leq T$ by $f \prec g \leq T$, for a convex g, enables efficient monitoring; alas, it might also result in potential false alarms (i.e., vectors u for which $f(u) \leq T$ but

[4]We deal here with differentiable functions, which include many functions of practical interest. Further, non-differentiable functions can be arbitrarily approximated by differentiable functions on any bounded domain.

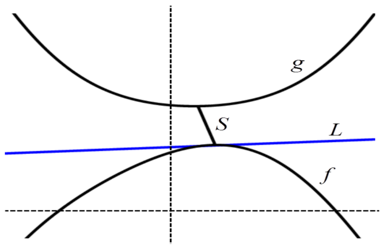

Fig. 6. A convex function g and concave function f such that $g > f$. S is the segment connecting the two closest points on the graphs. The tangent at the closest point on f's graph, L, satisfies $g > L > f$, proving that the set of f's tangent planes is a tight family of convex bounds.

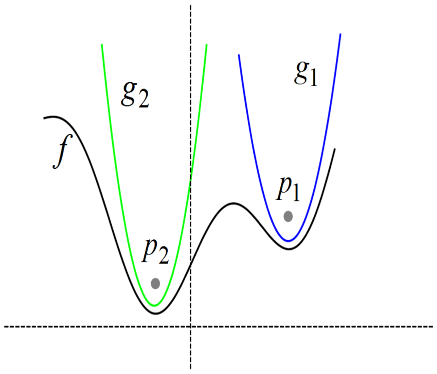

Fig. 7. The impossibility of choosing a single best convex bound for the function f (dark curve). g_1 (resp. g_2) is better in the vicinity of p_1 (resp. p_2).

$g(u) > T$). We refer to this problem as the *convexity gap*, or simply the gap (referring to the gap between f and g). Intuitively, the "system price" one must pay in order to allow distributed monitoring is reflected in the "convexity price," which is the gap between the monitored function and its convex bound. To minimize the number of false alarms, the gap should be minimized. However, as the following simple example demonstrates, it is often impossible to choose a single optimal g to achieve this goal. As depicted in Figure 7, it is evident that, loosely speaking, different bounds are better at different regions of the data space, and there is typically no hope of finding a unique bound that is *always* better than all the others. We formalize this observation with the following definition, which is both realizable and appropriate for the monitoring problem:

Definition 3.5. A convex bound g_1 is *better than g_2 at a point p* iff there exists a neighborhood of p in which $g_1 \prec g_2$.

Thus, in Figure 7, g_i is better at p_i for $i = 1, 2$.

Definition 3.5 is appropriate for the following reason. Recall that the local condition at the i-th node is $g(p_0 + d_i) \leq T$. Initially, the drift vector d_i is equal to zero; assuming that the data at the nodes behaves continuously or can be approximated by a random walk ([26, 34, 37, 38]), it follows that the local vector $p_0 + d_i$ can be modeled by a continuous process that starts at p_0 and gradually wanders away from it. Therefore, a bound is sought that is optimal (i.e., smaller than all other bounds) in a certain neighborhood of p_0. It turns out that in the general case, where f is neither convex nor concave, no such optimal bound exists (Lemma 3.6).

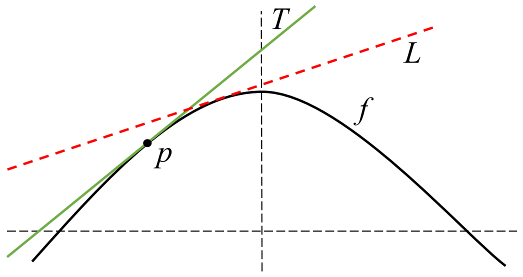

Fig. 8. A concave function f (dark), a point on the surface p (dark), a tangent plane at the point T (green), and a tangent plane at a different point L (dashed red). The tangent plane at the point bounds it tightly from above better than any other tangent plane. This holds in any dimension.

Lemma 3.6. *If f is neither convex nor concave, then there is no optimal convex upper bound g_{opt} such that $g_{opt} < g$ in some neighborhood of p_0 for every other convex upper bound g of f.*

Proof. Please see the Appendix, Section A.1 □

An optimal upper bound exists for the cases in which f is either convex (where the bound is f itself), or concave where the optimal upper bound is the tangent plane to f at p_0 (Lemma 3.7).

Lemma 3.7. *If f is concave, the tangent plane at a point p is the best convex bound at p.*

The proof is trivial. According to Lemma 3.4, if f is a concave function, then the tangent planes define a family of tight bounds. The tangent plane's value at p is equal to $f(p)$, but all other tangent planes lie above f. Thus, the deviation of the tangent plane at p from the point $(p, f(p))$ is quadratic; hence, locally, it is smaller than that of the tangent planes at other points, which is linear (see Figure 8).

Consequently, when bounding a concave function from above (or, equivalently, a convex function from below), we will replace it with its tangent plane at p_0. This is next used to transform a threshold condition on *general* functions to a convex condition.

3.4 "Convexizing" Threshold Conditions

If the monitored f is itself convex, the choice of a convex bound c is trivial—choose $c = f$. If f is concave, then, following Lemma 3.7, the tangent plane at p_0 is the optimal candidate for c. We next handle a more general case.

Definition 3.8. Assume that $f = c_1 - c_2$, where both c_1, c_2 are convex. The *convexization* of the condition $f \leq T$ is defined by $c = c_1 - L_{c_2}(p_0) \leq T$, where $L_{c_2}(p_0)$ is the linear approximation (tangent plane) of c_2 at p_0.

Since $c_1 - c_2 \leq T$ iff $c_1 \leq c_2 + T \triangleq c_3$, we can assume that the monitored condition is given as an inequality between two convex functions, $c_1 \leq c_3$. This condition is especially amenable to convexization: we replace it with $c_1 \leq L_{c_3}(p_0)$, where, as before, $L_{c_3}(p_0)$ is c_3's tangent plane at p_0. We will use this form of convexization for the inner product, cosine similarity, and PCA-Score functions (Section 4).

Note that the c defined in Definition 3.8 is convex, bounds f from above, and that its definition is motivated by the special cases where f is convex or concave. The lower bound case is similarly handled: the inequality $f \geq T$ is replaced by the condition $L_{c_1}(p_0) - c_2 \geq T$ (note that $L_{c_1}(p_0) - c_2$ is concave and bounds f from below).

We next prove that, for a very wide class of real problems, it is always possible to express f as the difference of two convex functions. First we recall a definition from calculus that comes in handy for testing convexity:

Definition 3.9. Let f be a function of $x_1 \ldots x_n$. Its *Hessian* H_f is the $n \times n$ matrix $H_f(i,j) = \frac{\partial^2 f}{\partial x_i \partial x_j}$.

It is well known that a function is convex in a given domain D iff its Hessian is positive semi-definite (PSD)[5] at every point in D.[6] This is a generalization of the single variable case, where a function is convex in a given domain D if its second derivative is positive.

Lemma 3.10. *If f possesses bounded second derivatives in a domain D, it can be expressed as the difference of two convex functions.*

Proof. Since the elements of H_f are bounded over D, there is an upper bound, Λ, on the absolute values of H_f's negative eigenvalues. Define $c_2(u) = \frac{\Lambda}{2}\|u\|^2$, $c_1(u) = f(u) + c_2(u)$. Clearly, $f = c_1 - c_2$ and c_2 is convex (its Hessian H_{c_2} is positive definite). Also, $H_{c_1} = H_f + H_{c_2} = H_f + \Lambda I$ (where I is the identity matrix). Hence, all the eigenvalues of H_{c_1} are greater than 0 and c_1 is convex. □

Example 3.11. To understand the intuition behind Lemma 3.10, assume we want to express the single variable function $f(x) = \sin(x)$ as a difference of convex functions (to "convexize" it). Since the absolute value of its second derivative is bounded by $\Lambda = 1$, we define $c_2(x) = \frac{1}{2}x^2$, $c_1(x) = \sin(x) + \frac{1}{2}x^2$ (constructed as in the proof). $c_2''(x) = 1 \geq 0$, so $c_2(x)$ is convex. $c_1''(x) \geq 0$ since $c_1''(x) = f''(x) + c_2''(x)$ and $f''(x) \geq -1$, meaning that $c_1(x)$ is also convex. Writing $f(x)$ as the difference of two convex functions we get $f(x) = (\sin(x) + \frac{1}{2}x^2) - \frac{1}{2}x^2$.

All the functions we deal with in this article either have bounded second derivatives or their derivatives are continuous and the domain of interest is bounded; hence, Lemma 3.10 is applicable. We will apply it for monitoring cosine similarity (Section 4.3).

3.5 Range of Applicability and Limitations

The proposed CB method can in principle be applied to any problem that can be posed as functional threshold monitoring over a distributed system, provided that the monitored function can be evaluated at the average of locally defined vectors at the nodes. Crucially, not just the function, but these "local vectors" may contain non-linear elements, which very significantly broadens the scope of applications.

The GM paradigm (which includes CD), operating under the same assumptions, has been successfully applied to a myriad of problems, including, recently, the monitoring of linear regression [19], entropy [20], properties of large, distributed graphs [66], and AMS sketches [22, 42], in addition to earlier work on the chi-square coefficient, inner product, cosine similarity, and more. Recall that every solution that GM provides is also realizable by CB (see Theorem 3.1),

For a function to be amenable for the CB method, it must be either convex, or more generally, equal to the difference of two convex functions, as explained in Section 3.4. The family of functions that can thus be represented is very broad (see Lemma 3.10). If the function is continuous but not differentiable, it can still be approximate to an arbitrary degree by polynomials, as follows from the celebrated *Stone-Weierstrass Theorem*, [61], which are, of course, differentiable to arbitrary order, hence Lemma 3.10 can be applied to them; see also Reference [1].

One could, of course, envision functions that are very difficult to monitor using CB: as an extreme example, assume that the local values at the nodes are infinite precision real numbers, and

[5]A matrix B is PSD iff $uBu^t \geq 0$ for every vector u. A symmetric matrix is PSD iff all its eigenvalues are ≥ 0.
[6]For a comprehensive study of convexity, see Reference [8].

one wishes to monitor the Dirichlet function $D(x)$, which obtains a value of 1 at the rational numbers and 0 at the irrational ones. This function is discontinuous at every point, while it is well-known that a convex function (hence the difference of two convex functions) can be discontinuous only on a countable set (see Chapter 3.14 in Reference [65]); hence $D(x)$ cannot be expressed as as difference of convex functions. However, it is quite unclear how *any* algorithm can monitor such pathological functions effectively; moreover, while interesting from the purely theoretical point of view, it is difficult to see how such functions can appear in practical monitoring problems.

A somewhat less extreme example are functions that are continuous but oscillate very quickly, e.g., a function of one variables x defined by $f(x) = \sin(Mx)$, where M is very large [59]. If the threshold T is between 0 and 1, then very small changes in x will cause a violation; thus, every monitoring algorithm will run into difficulties when handling $f(x)$. In the GM approach, this difficulty will be manifest in very small safe-zones; with CB, the pathological nature of $f(x)$ will be manifest in a very large convex bound, since the convexization based on Lemma 3.10 will express $f(x)$ as $(\sin(Mx) + \frac{M^2}{2}x^2) - \frac{M^2}{2}x^2$ (compare to the example 3.11). Thus GM and CB can be applied to such functions, but they will perform poorly in terms of communication.

Recall that to express a function as a difference of convex functions we need to bound from below the negative eigenvalues of the Hessian matrix (proof of Lemma 3.10). Finding a bound may pose a difficulty since the bound should hold for the entire domain of interest, and not just one point. Moreover, if the bound on the eigenvalues is not tight, the convex bound would be ”loose”, leading to extra communication due to false violations. Bounding the eigenvalues can be difficult, but as our experience proved, symbolic computation software (such as Wolfram and Maple) can assist in deriving such bounds. Alternative techniques, as in Reference [1], that study the problem of representing polynomials as a difference of convex functions, should be further explored. The problem of developing methods to quickly find convex bounds that can be efficiently computed and also yield a small number of false alarms is not entirely solved, and we expect it to be an active, interesting field of research.

4 APPLYING CB

We next apply CB to monitor four popular functions: the PCC, inner product, cosine similarity, and PCA-Score (“effective dimension”). The PCC is often used to measure correlation between binary events (please see an example application of air quality monitoring in the Introduction). Inner product and cosine similarity are similarity metrics. They can be used to measure the similarity between multi-dimensional vectors. For example, one may be interested to know the (dis)similarity between search terms used by different communities or the (dis)similarity between twitter hashtags used by different social groups. The search terms or hashtags (in a specific time window) can be arranged as vectors, and the inner product of these vectors can be calculated to give a size-dependent score of the similarity. On the other hand, cosine similarity can be used for a normalized similarity score. The PCA score captures the effective dimension of a PCA matrix. As noted by Lakhina et al. [39] and others, some systems (or environments) have an intrinsically low effective dimension. A change in the effective dimension signifies system health issues or a phase change (such as a sharp model drift).

These functions were chosen both for their great practical importance and since they do not fall into any category for which there exist simple, efficient solutions (they are not linear, convex, concave, or monotonic). In Sections 6, 7, and 8, we compare the runtime, communication overhead, and power consumption of CB and GM in a variety of real scenarios.

To apply CB, we follow the method described in Section 3.4. If the monitored function cannot be directly written as the difference of two convex functions (as in the case of cosine similarity), we apply Lemma 3.10.

4.1 PCC

The PCC measures the linear correlation between two variables. Let x, y denote the frequency of appearances of two items in elements of a certain set, and z the frequency of their joint appearances. A very typical example is when x, y denote the ratio of documents in which certain terms appear, and z the same for appearances of both terms simultaneously. The range over which PCC is defined is therefore $0 \leq x, y \leq 1$ and $z \leq x, y$. The function measures the strength of correlation between the appearances of x and y, and is defined by

$$P(x,y,z) = \frac{z - xy}{\sqrt{x - x^2}\sqrt{y - y^2}} \quad . \tag{1}$$

We will assume $T > 0$; the case $T \leq 0$ is treated similarly.

The condition $P(x,y,z) \leq T$ can be written as $z \leq xy + T\sqrt{x-x^2}\sqrt{y-y^2}$. We convexize it as follows. First, note that xy is neither convex nor concave; it is trivial to verify that the Hessian's eigenvalues for xy are always 1 and −1 (i.e., every point on the function's surface is a *saddle point*). We therefore use the identity $xy = \frac{(x+y)^2}{4} - \frac{(x-y)^2}{4}$. Denote $Q_1 = \frac{(x+y)^2}{4}, Q_2 = \frac{(x-y)^2}{4}$ (note that Q_1, Q_2 are convex). We also need the following:

Lemma 4.1. *The function* $\sqrt{x-x^2}\sqrt{y-y^2}$ *is concave.*

Proof. Please see the Appendix, Section A.3. □

The condition $P(x,y,z) \leq T$ can therefore be written as

$$(z - T\sqrt{x-x^2}\sqrt{y-y^2} + Q_2) - Q_1 \leq 0, \tag{2}$$

and, since this last expression is the difference of two convex functions,[7] we can proceed by applying the paradigm described in Definition 3.8. It remains to calculate the tangent planes of Q_1 (for the upper bound), and Q_2, and $G = \sqrt{x-x^2}\sqrt{y-y^2}$ (for the lower bound) at (x_0, y_0). We start by calculating the gradients of the respective functions:

$$\begin{aligned}
\nabla Q_1(x_0,y_0) &= 0.5(x_0 + y_0, x_0 + y_0), \\
\nabla Q_2(x_0,y_0) &= 0.5(x_0 - y_0, y_0 - x_0), \\
\nabla G(x_0,y_0) &= \left(\frac{(1-2x_0)\sqrt{y_0 - {y_0}^2}}{2\sqrt{x_0 - {x_0}^2}}, \frac{(1-2y_0)\sqrt{x_0 - {x_0}^2}}{2\sqrt{y_0 - {y_0}^2}}\right).
\end{aligned}$$

Next we use them to calculate tangent planes by plugging them into the Taylor expansion. Recall that the tangent plane of a multivariate function $f(u)$ at u_0 is given by its first-order Taylor expansion $f(u_0) + \langle \nabla f(u_0), u - u_0 \rangle$.

Finally, the convex local condition for monitoring the upper bound is given by

$$z - G(x,y) + Q_2(x,y) - (Q_1(x_0,y_0) + \langle \nabla Q_1(x_0,y_0), (x - x_0, y - y_0)\rangle) \leq 0,$$

where Q_1 of the original condition was replaced by its tangent plane; the lower bound is similarly handled (by replacing Q_2 and G with their tangent planes). As long as this condition holds, it is guaranteed that the global lower bound is not breached. The convex upper bound and concave upper bound are depicted for some representative values in Figure 9.

[7]Since $T\sqrt{x-x^2}\sqrt{y-y^2}$ is concave, its negative is convex.

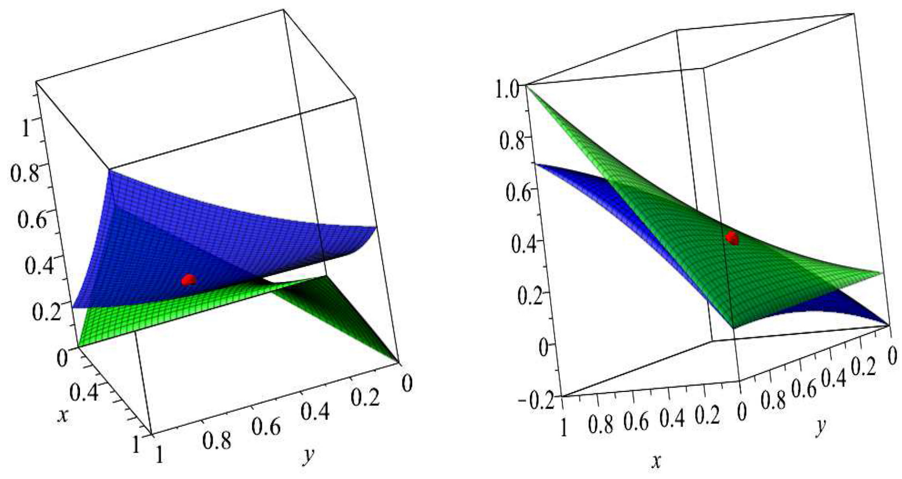

Fig. 9. Left: a convex upper bound (blue) for PCC (green). The reference point (in red) is $x_0 = 0.3, y_0 = 0.6$, and $T = 0.4$. Right: a concave lower bound.

4.1.1 Monitoring PCC with GM. As explained in Section 2.2, to apply GM we must be able to solve the closest point problem for the surface defined by $z = xy + T\sqrt{x - x^2}\sqrt{y - y^2}$. To this end, we used dedicated software [27], which first reduces the surface's equation to an algebraic one and then solves for the closest point. This incurred a runtime far higher than the simple CB solution (by more than three orders of magnitude) and also resulted in higher communication overhead; results are provided in Section 6.1.

4.2 Inner Product

The inner product function is extensively applied in data mining and monitoring tasks as a measure of similarity. We assume that the monitored function f is over vectors of length $2n$, and is equal to the inner product of the first and second halves of the vector; denoting the concatenation of vectors x, y by $[x, y]$, we have $f([x, y]) = \langle x, y\rangle$. To express f as the difference of two convex functions, note that $4\langle x, y\rangle = \|x + y\|^2 - \|x - y\|^2$. Since the norm squared function is convex, the condition $\langle x, y\rangle \le T$ is convexized by

$$\|x + y\|^2 \le 4T + \|x_0 - y_0\|^2 + [x_0 - y_0, y_0 - x_0], [x - x_0, y - y_0], \tag{3}$$

where the reference point $p_0 = [x_0, y_0]$, and the gradient of $\|x - y\|^2$ is equal to $2[x - y, y - x]$ (recall that, for a multivariate function f, the tangent plane at a point u_0 is given by $f(u_0) + \langle \nabla f(u_0), u - u_0\rangle$).

4.2.1 Monitoring Inner Product with GM. To apply GM, one must be able to solve the closest point problem for the threshold surface, $\langle x, y\rangle = T$. If the point outside the surface is denoted $[x_0, y_0]$, the problem can be formulated as

$$\text{Minimize } (\|x - x_0\|^2 + \|y - y_0\|^2) \text{ such that } \langle x, y\rangle = T.$$

This problem can be solved with Lagrange multipliers. Define
$F \triangleq \|x - x_0\|^2 + \|y - y_0\|^2 + 2\lambda(\langle x, y\rangle - T)$. The equations $\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial \lambda} = 0$ assume the form

$$(x - x_0) + \lambda y = 0,\ (y - y_0) + \lambda x = 0,\ \langle x, y\rangle = T. \tag{4}$$

These equations can be solved by first extracting x, y as functions of x_0, y_0 from the first two equations and then plugging the result into the third equation, $\langle x, y\rangle = T$. Skipping the details, this yields a quartic equation in λ:

$$T\lambda^4 - (2T + \langle x_0, y_0\rangle)\,\lambda^2 + \left(\|x_0\|^2 + \|y_0\|^2\right)\lambda - \langle x_0, y_0\rangle + T = 0.$$

After solving for λ, it is easy to solve for x, y.

While the inner product case is the only one addressed here for which a relatively simple solution for GM could be found, it still incurs the overhead of computing the quartic's coefficients, solving it, and checking the solutions to see which one yields the closest point on the surface. GM's runtime was about five times higher than CB's.

4.3 Cosine Similarity

Another very popular measure of similarity is *cosine similarity* (referred to as *Csim* hereafter), which resembles the inner product function, but normalizes it by the length of the vectors. For example, if we have two histograms of word frequencies, derived from two document corpora, Csim will "neutralize" the effect of the corpus size when measuring the histogram similarity; the inner product function, however, is biased toward larger corpora.

As in the inner product case, the data vector p is $[x, y]$, the concatenation of two n-dimensional vectors x, y, and the reference point will be denoted $p_0 = [x_0, y_0]$. Then, Csim is defined by $\text{Csim}(p) = \frac{\langle x,y\rangle}{\|x\|\|y\|}$. Thus, to monitor a lower bound, i.e., $\text{Csim}(p) \geq T$ (we assume $T > 0$; the case of negative T is similarly treated), we need to monitor the condition $\langle x, y\rangle \geq T\|x\|\|y\|$. This problem is more complicated than the inner product case, since there is no obvious way to decompose it into an inequality between two convex functions; this is because while representing $\langle x, y\rangle$ as the difference of two convex functions is relatively easy, it is more difficult to derive such a representation for $\|x\|\|y\|$. We therefore resort to using the method outlined in Lemma 3.10. We must first determine the smallest eigenvalue of the Hessian of $\|x\|\|y\|$. It follows from the following lemma that it equals -1:

Lemma 4.2. *At a point x, y, the eigenvalues of $H(\|x\|\|y\|)$ are $1, -1$ (each with multiplicity one) and $\|x\|/\|y\|$, $\|y\|/\|x\|$ (each with multiplicity $n-1$).*

Proof. Please see the Appendix, Section A.4. □

Now we can proceed to convexize the problem. First, we write the inequality $\langle x, y\rangle \geq T\|x\|\|y\|$ as $\|x + y\|^2 \geq \|x - y\|^2 + 4T\|x\|\|y\|$. Next, to make both sides convex, we add $2T(\|x\|^2 + \|y\|^2)$ to them, to obtain

$$\|x + y\|^2 + 2T(\|x\|^2 + \|y\|^2) \geq \|x - y\|^2 + 4T\|x\|\|y\| + 2T(\|x\|^2 + \|y\|^2).$$

The inequality is convexized by replacing the LHS with its tangent plane at $p_0 = [x_0, y_0]$. Denoting the LHS by $h([x, y])$ and the RHS by $g([x, y])$ the convexized condition is given by

$$h([x_0, y_0]) + \langle\nabla h([x, y]), [x - x_0, y - y_0]\rangle \geq g([x, y]),$$

where $\nabla h([x, y]) = 2[x + y + 2Tx, x + y + 2Ty]$.

4.3.1 Monitoring Csim with GM. The problem of calculating the distance of a point to the Csim surface $\{[x, y] | \langle x, y\rangle = T\|x\|\|y\|\}$ is exceedingly difficult and no closed-form solution exists. We were able to simplify the computation of the closest point, reducing it to an optimization problem in merely three variables regardless of the dimensions of x, y. Nonetheless, three different software packages we applied took about 3 minutes to complete the task for a *single* point.

Details on the closest point problem for Csim are in the Appendix, Section A.5.

4.4 PCA-Score

PCA (principal component analysis) is a fundamental dimension reduction technique with numerous applications. Given a set of vectors in Euclidean space, PCA seeks a low-dimensional subspace which, on the average, well-approximates the vectors in the set. Formally:

Definition 4.3. Given $1 > R > 0$ (typically $R \approx 0.9$) and a finite set of vectors $S \subset \mathcal{R}^m$, the *effective dimension* of S is defined as the smallest dimension of a sub-space $V \subset \mathcal{R}^m$, such that $\sum_{u \in S} \|P_V(u)\|^2 \geq R \sum_{u \in S} \|u\|^2$, where $P_V(u)$ is the projection of u on V.[8]

It is well known that the effective dimension, denoted k hereafter, can be computed as follows:

(1) Construct the $m \times m$ *scatter matrix* $M = \sum_{u \in S} uu^t$. Note that in the distributed setup, S is equal to the sum of local scatter matrices at the nodes.
(2) Compute M's eigenvalues, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$.
(3) Determine the smallest k such that $\sum_{1 \leq i \leq k} \lambda_i^2 \geq R \sum_{1 \leq i \leq m} \lambda_i^2$.

In Reference [39], PCA is applied to measure the health of a system consisting of distributed nodes: at each timestep, a vector of various system parameters is associated with each node (typically, the vectors' components are various traffic volume indicators). System-wide anomalies (e.g., DDOS attacks) are highly correlated with an increase in the effective dimension of the union of the parameter vectors over all nodes.

Given an initial effective dimension k for some $1 > R > 0$, we define the *PCA-Score* by $P_k = (\sum_{1 \leq i \leq k} \lambda_i^2)/(\sum_{1 \leq i \leq m} \lambda_i^2)$. Initially k is chosen such that $P_k \geq R$, as the stream progresses the effective dimension may change: $P_k < R$ indicates an increase in the effective dimension (relative to the initial k). Hence, we wish to monitor the PCA-Score and alert when it drops below the threshold $T = R$.

As for the previous functions we handled, the difficulty lies in that λ_i are the eigenvalues of a global matrix equal to the sum of the local matrices; hence, its exact computation at every timestep will incur a huge communication overhead. To apply CB for distributed monitoring, we must express the PCA-Score as a function of the average matrix, as opposed to the sum; however, since (i) eigenvalues scale linearly when the matrix is multiplied by a scalar, and (ii) the PCA-Score is defined as the ratio of sums of squares of eigenvalues, its values on the average and sum matrices are equal.

What remains is to "convexize" the inequality

$$\sum_{1 \leq i \leq k} \lambda_i^2 \geq T \sum_{1 \leq i \leq m} \lambda_i^2. \tag{5}$$

We rely on the following two lemmas:

Lemma 4.4. *For an $m \times m$ scatter matrix M, $\sum_{1 \leq i \leq m} \lambda_i^2$ equals $\mathrm{Tr}^2(M)$ and is a convex function of M. The proof follows immediately from the fact that every scatter matrix is symmetric.*

Lemma 4.5. *For a symmetric M, $\sum_{1 \leq i \leq k} \lambda_i^2$ is convex.*

Proof. This follows from the well-known *Fan identities*, specifically Theorem 2 in Reference [17]. □

Since both sides of Equation (5) are convex, we can proceed as in Section 3.4, by replacing the LHS with the tangent plane at the reference scatter matrix M_0. All that is required is to compute the gradient of the LHS; for that, we use the following result from linear algebra.

[8] We assume that S is *centralized*, i.e., its average is zero. The general case proceeds along the same lines.

LEMMA 4.6. *The derivative of λ_i with respect to M is equal to $e_i e_i^t$, where e_i is the eigenvector of M corresponding to λ_i.*

Hence, the monitored condition in Equation (5) is convexized by

$$\sum_{1\leq i\leq k} \lambda_i^2(M_0) + 2\left\langle \sum_{1\leq i\leq k} \lambda_i(M_0)e_i(M_0)e_i^t(M_0), M - M_0 \right\rangle \geq T(\mathrm{Tr}(M^2)), \tag{6}$$

where M_0 is the reference matrix and M the local matrix.

4.4.1 Monitoring PCA-Score with GM. To apply GM, we must be able to compute the minimal PCA-Score over all matrices in a sphere in the m^2-dimensional space of $m \times m$ matrices. This can be done using the *perturbative bounds* applied in Reference [29], which also addressed monitoring the health of a distributed system. We have also tested a simpler method, analogous to the ones used in Reference [29], in which the local condition is defined as containment in the maximal sphere around the reference matrix that is contained in the admissible region. Both methods require bounding the change in the eigenvalues, given the magnitude of change in the matrix. Two such perturbative bounds can be applied, which relate the change in the eigenvalues to the *Frobenius norm* (FN) or the *spectral norm* of the change in the matrix (i.e., drift vector). We refer to the algorithms that use the *FN* resp. *spectral norm (SN)* as FN resp. SN. For better readability, these methods are described in an appendix (Section A.2); see also Reference [8]. We note that all these methods (GM, FN, SN) require solving the difficult problem of finding the closest point on the surface of matrices whose PCA-Score equals T; this renders them slower than CB. Further, CB was better at reducing communication overhead. Details are provided in Section 6.4.

5 EXPERIMENTAL SETTINGS

In the experiments, CB was compared to GM for the tasks of monitoring the functions discussed in Section 4, over a few datasets and for different threshold values. To the best of our knowledge, GM represents the state-of-the-art in monitoring threshold queries over distributed streams. We are not aware of any other work on monitoring cosine similarity and the PCC, and while there is other work on monitoring the inner product [13], GM improved on it [22]. For the PCA-score function, we also compared CB to the FN and SN perturbative bounds described in Reference [29].

We examined the sliding window scenario, in which the data of interest are the last m records for some predefined m (or the last records received within a certain period); for example, one may wish to continuously monitor only the last 1,000 tweets in a tweet stream. The sliding window case corresponds to the *turnstile model*, in which the data vector's entries can both increase and decrease, and is more general than the *cash register* model, in which the entries can only increase.

In all the experiments, CB outperformed the other methods in both communication reduction and runtime, with the improvement factor in runtime being orders of magnitude for PCC, cosine similarity, and PCA-Score.

We also performed experiments to evaluate the power consumption of CB vs. GM on two resource-limited devices: a VOYO Mini-Pc and an Edison SoC. As expected, the experiments show that CB's advantage in runtime is translated to an advantage in power consumption. CB's power consumption was much lower than GM's, reaching orders of magnitude for most functions, making it more suitable for battery operated devices. Details on setup and results are in Section 8.

5.1 Data

We used three real datasets: the Reuters Corpus (RCV1-v2, REU), a Twitter crawl (Dataset-UDI-TwitterCrawl-Aug2012, TWIT), and the 10% sample supplied as part of KDD Cup 1999 Data (KC). The overall sizes of these datasets were REU 374MB, TWIT 691MB, KC 46MB.

REU consists of 804,414 news stories, produced by Reuters between August 20, 1996, and August 19, 1997. Each story was categorized according to its content and identified by a unique document ID. REU was processed by Lewis et al. [43]. A total of 47,236 features were extracted from the documents and then indexed. Each document is represented as a vector of the features it contains. We simulate 10 streams by arranging the feature vectors in ascending order (according to their document ID) and selecting feature vectors for the streams in round-robin fashion.

TWIT is a subset of Twitter, containing 284 million follower relationships, 3 million user profiles, and 50 million tweets. The dataset was collected during May 2011 by Li et al. [45]. We filtered the dataset to obtain only hashtagged tweets, which left us with 9 million tweets from 140,000 users. For each tweet, the dataset contains information about the tweet content, tweet ID, creation time, re-tweet count, favorites, hashtags, and URLs.

KC was used for the Third International Knowledge Discovery and Data Mining Tools Competition. The original task was to build a network intrusion detector. The dataset contains information about TCP connections. Each connection is described by 41 features, including duration, protocol, bytes sent, bytes received, and so forth.

The REU and TWIT datasets contain features from textual data, and they were used to evaluate the cosine similarity and inner product functions, which are common tools in text mining and analysis. The PCC function, measuring the correlation between feature appearances, was evaluated using the REU dataset. The TWIT dataset was not used to evaluate the PCC function, because no hashtag (and no hashtag pair) appear in the stream frequently enough to make the monitoring task worthwhile. The KC dataset, containing vectors of numerical features describing TCP connections, was used to evaluate the PCA Score function (which has been used in system monitoring context [39]).

For all datasets, to simulate multiple streams, we distributed the data between the nodes in round-robin fashion. Results are presented for 10 streams, and in Section 7 we present some results for communication reduction for up to 1,000 streams (the reduction in computational overhead does not depend on the number of streams).

6 COMPUTATIONAL OVERHEAD REDUCTION

Next we discuss the main results of this article—the reduction in running time for monitoring the four functions presented in Section 4. In the following sections, we discuss the communication reduction results and the evaluation of power consumption on resource-limited devices.

In Figure 10, we present a summary of the running times for GM and CB, on the various functions and datasets. In all cases CB outperforms the previous state-of-the-art. Per function details are provided in Sections 6.1 to 6.4.

6.1 Pearson Correlation Coefficient

We evaluated PCC on REU, where every document may be labeled as belonging to several categories. The most frequent category is CCAT (the CORPORATE/INDUSTRIAL category). In the experiments, our goal was to select features that are most relevant to this category, i.e., whose PCC with the category is above a given T. Each node holds a sliding window containing the last 6,700 documents it received (this is roughly the number of documents received in a month). We monitored the correlation of CCAT with the features Bosnia and Febru. We chose these two features because they display a different characteristic behavior. The PCC value for the feature Bosnia displays a gradually inclining trend as the stream evolves (see Figure 19), while the PCC value for the feature Febru displays a sharper decline at first, which becomes an incline about halfway through the stream (see Figure 17).

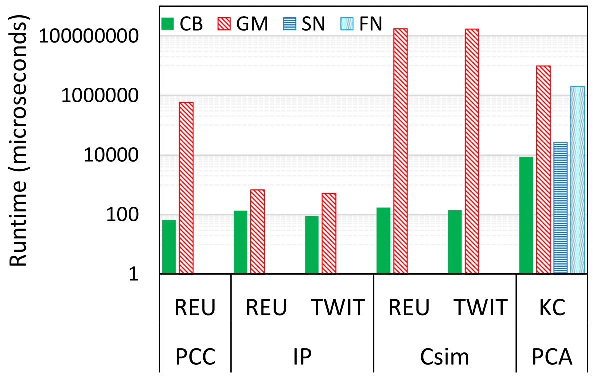

Fig. 10. Running times for a local condition check. SN and FN stand for previous methods to monitor PCA-Score (see Section 6.4). CB is the fastest method (by orders of magnitude faster in most cases). *Note the logarithmic scale.*

Table 1. Runtime for Checking the Local Condition Using CB vs GM—Even for the Inner-Product Function Where No Optimization Is Required, CB Is Considerably Faster

			Runtime (seconds)		
Function	Dataset	Dim	GM	CB	Speedup
PCC	REU	3	0.58	0.67E-04	8,657.7
Inner-Prod	REU	4100	1.35E-04	6.89E-04	5.10
Inner-Prod	TWIT	2500	8.90E-05	5.20E-04	5.84
Csim	REU	4100	1.73E-4	175	1,000,000
Csim	TWIT	2500	1.41E-04	170	1,200,000

Runtime Evaluation. The majority of GM's runtime is spent on testing for sphere intersection with the PCC surface. To solve this problem we used the Gloptipoly global optimization package [27]. In CB, the local conditions for PCC monitoring are very simple; they only require computing the functions composing the PCC and their derivatives (Section 4.1).

The experiments demonstrated that the runtime of checking the local condition for the CB method is almost four orders of magnitude lower than for GM (see Table 1). *Note*: The table also include results for inner product and Csim.[9]

6.2 Inner Product

We monitored the inner product on REU and TWIT. As in Reference [22], we calculated the inner product of feature vectors from two streams (created by splitting the records). For REU, we used the top 2,050 features left after removing features that appear in less than 1% of the documents. We used the NLTK [7] package to tokenize and stem the tweets in TWIT and then selected the top 1,250 features, ignoring features appearing in less than 0.1% of the tweets.

In the REU experiment, each node held a sliding window of the last 6,700 documents, while in TWIT each node held a sliding window containing the last 1,000 tweets. We used threshold values between 7,000 and 17,000 for TWIT, and between 4.9E7 to 5.5E7 for REU.

[9]In the PCA-Score experiments (Section 6.4), we compared CB to three different methods; hence the results are provided separately in Table 2.

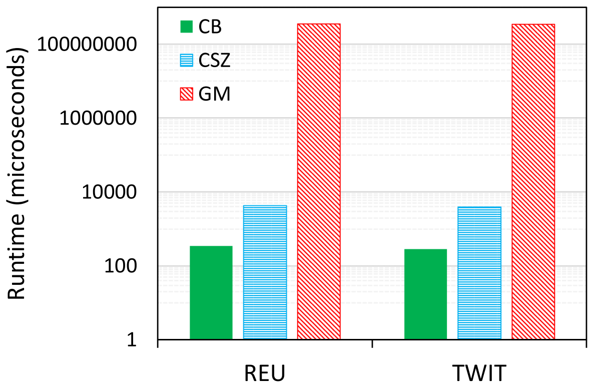

Fig. 11. Running times for a local condition check. While the decomposition method (CSZ) is much faster than GM, CB is the fastest of the three. *Note the logarithmic scale.*

Runtime Evaluation. Although GM requires no optimization to find the closest point on the surface but only to solve a quartic equation, CB checks local conditions about five times faster (see Table 1); this is due to the time required to construct and solve the equation, and then to check the distinct solutions to see which one yields the closest point. Checking the local conditions requires more time for the REU dataset, since the feature vectors are longer (2,050 vs 1,250).

6.3 Cosine Similarity

To evaluate the computational overhead for the cosine similarity function, we monitored both REU and TWIT, using the same settings as the inner product experiments (see Section 6.2).

Runtime Evaluation. CB was about six orders of magnitude faster than GM for both datasets. The run-time of a local condition check in GM is almost 3 minutes, while for CB it is less than 0.2 milliseconds (see Table 1). This is not entirely surprising, as the closest point problem for the cosine similarity function is very difficult; see Section 4.3.1.

6.3.1 Comparison with CSZ. Recently, Lazerson et al. [40] proposed the *CSZ* method, which can be used for tracking a complex function by decomposing it into simpler primitives and tracking them simultaneously. As an example, they decomposed the cosine similarity function and tracked it using CSZ. They noted that a direct application of the CB method yields lower communication costs than CSZ; however, they did not evaluate running times, which are the main focus of this work. Our goal here is to compare the runtime of CB to that of CSZ, where each (simple) primitive is tracked using GM.

Following Lazerson et al. [40], we decompose cosine similarity into three simpler primitives: the inner product $\langle x, y\rangle$ and the norms of x and y $\|x\|$, $\|y\|$. Note that the distance to the threshold surface defined by each of these functions (and therefore the local condition) can be computed using a closed form solution and requires no optimization (see Section 4.2 for the inner product, and Reference [40] for the norm).

We tracked the cosine similarity function on REU and TWIT using a relative approximation bound of 0.1 and compared CB to CSZ. Figure 11 shows the runtimes of CB and CSZ for the two datasets. For completeness, we also include the runtime of a direct application of GM. GM has the worst runtime by far; CSZ is almost five orders of magnitude better, but it is still an order of magnitude slower than CB despite all the functions it tracks having closed-form solutions. The communication cost of CB is superior to that of CSZ (see also Reference [40]). In our evaluation,

Table 2. Runtimes (Seconds) for Monitoring the PCA-Score Over KC

	CB	SN	FN	GM
runtime	0.0086	0.027	2.01	9.57
CB speedup	1	3.20	232.81	1105.95

the communication incurred by CB was about 3.6 times lower than CSZ for the TWIT dataset and about 10 times lower for REU.

6.4 PCA-Score

For monitoring the PCA-Score function, we compared CB with GM as well as with methods based on the FN and SN perturbative bounds, described in Reference [29] (see also Section 4.4). All methods except CB require solving complex optimization problems, which were implemented using Matlab and the CVXOPT[10] package.

We monitored the PCA-Score over KC using 10 nodes, each holding a sliding window of the last 100 feature vectors. We ran experiments with threshold values T ranging between 0.8 and 0.95, and effective dimension values ranging from 3 to 6.

Runtime Evaluation. Runtime results for monitoring the PCA-Score are displayed in Table 2. The table shows the runtime of a local condition check of each method as well as the speedup factor achieved by CB.

CB is about three times faster than SN, two orders of magnitude faster than FN, and three orders of magnitude faster than GM. Note that while SN runtime results are better than GM's, it achieves a rather poor reduction in communication (Figure 13).

Runtime/Communication Trade-off. Since here we evaluate four methods (CB, GM, SN, FN), and not just to CB and GM, we wish to present a unified view of how all methods perform in terms of both communication cost and runtime. The experiments show that the three previous methods that were compared with CB—SN, FN and GM—offer a tradeoff between communication cost and runtime.

GM achieves the best communication cost of the three but is also the slowest method. FN is faster than GM but its communication cost is slightly higher. SN is the faster of the three by far, but it achieves relatively poor communication reduction. CB improves on all three methods, achieving better communication cost than GM and better runtime than SN (Figure 12).

7 COMMUNICATION OVERHEAD REDUCTION

While the focus of the CB method was on reducing computational overhead, we also wanted to evaluate its impact on the communication cost. Clearly, reducing computation cost at the price of a large communication overhead is unacceptable. Our evaluation shows that CB does not incur extra communication costs above GM. In fact, CB offers a modest improvement in communication overhead (in addition to the considerable improvement in runtime). In this section, we provide results on the performance of CB in reducing overall communication.

To evaluate the communication cost, we measured the number of messages sent. The naive method, in which every message is sent to the coordinator, is used as a common baseline, and communication cost is reported using *ratio to naive*. At the opposite extreme, we compared to a hypothetical algorithm, which alerts only when the threshold condition is locally violated, i.e.,

[10]http://cvxopt.org/.

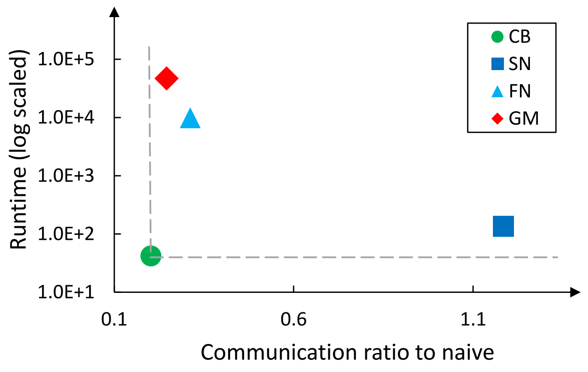

Fig. 12. Runtime and communication costs of the different methods used to monitor the PCA-Score. The previous methods—SN, FN and GM—offer a tradeoff between speed and efficiency. CB dominates them, offering both faster runtime and better communication cost.

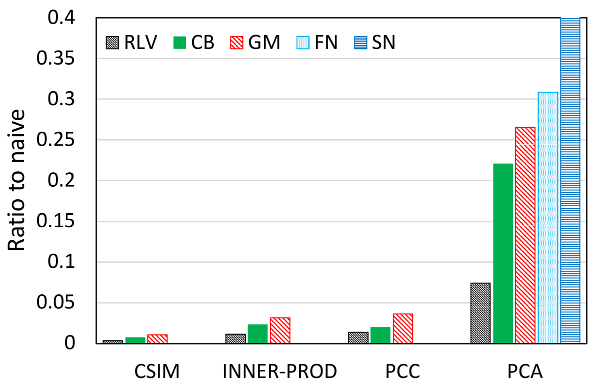

Fig. 13. Communication reduction summary (lower is better). CB's communication cost is lower than that of the other methods. In most cases, it is very close to the super-optimal lower bound (RLV).

when $f(v_i) \geq T$ for some local vector v_i. Clearly, every monitoring algorithm will have to alert in such a case. However, to maintain correctness, the local conditions have to adhere to the more restrictive constraint $f(\frac{v_1+\ldots+v_k}{k}) \leq T$. Since the constraints of every correct algorithm are more restrictive, it will issue more alerts, leading to a higher communication cost (unless, of course, f is convex). We refer to this super-optimal bound—the number of local violations–as RLV (real local violations); if the ratio between the number of messages sent by a certain algorithm and the number RLV sent is close to 1, then hardly any additional improvement is possible.

Figure 13 shows a summary of the communication required by CB, GM, and RLV for the functions we studied as well as SN and FN for the PCA-Score function. Each bar represents the results across multiple thresholds and datasets. CB is always better than GM. In most cases CB is close to the super-optimal lower bound RLV, meaning that hardly any improvement is possible. Note that while FN and SN displayed better runtimes than GM (Table 2), they have higher communication costs. CB is better than the competing methods (GM, FN, SN) in both runtime and communication costs (see also Figure 12).

Testing the effect of the window size on the communication cost revealed that communication cost decreases as the window size grows; see Figure 14. This decrease is due to the slower change

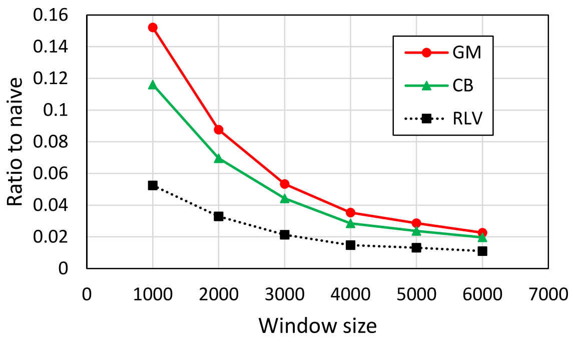

Fig. 14. Relative communication cost as a function of window size for inner-Prod on TWIT (lower is better). As window size increases, the communication cost drops.

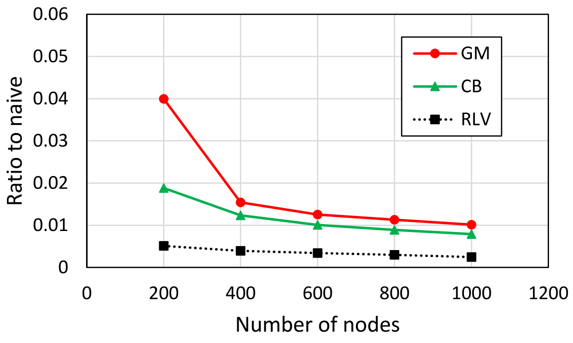

Fig. 15. Scalability with the number of nodes (inner-prod, TWIT). Relative communication cost for up to 1,000 nodes (lower is better). The improvement factor over the naive method grows with the number of nodes.

in the function's value. This explains the relatively modest communication reduction for the PCA function (Figure 13), where a small window was used.

To test the scalability of CB, we ran experiments with up to 1,000 nodes. Figure 15 shows the results. The advantage of both CB and GM (and RLV) over the naive method grows with the number of nodes, while CB maintains its superiority over GM.

Our evaluation across multiple datasets using various threshold values showed that CB's communication cost was better than GM's not only on the average but for all cases we tested. It is also evident that both CB and GM are affected by the choice of threshold values regardless of the specific function or dataset. Threshold values that are crossed more often cause higher communication overhead for both methods (since the monitoring task is more difficult).

We next provide a detailed study and analysis of CB's improvement over GM in reducing communication overhead for each of the functions.

7.1 Pearson Correlation Coefficient

CB performed better than GM for all thresholds (Figures 16 and 18). The advantage when monitoring Bosnia was larger, with CB typically performing two to three times better. This is due to there being much less room for improvement for Febru, as indicated by the proximity to the RLV bound.

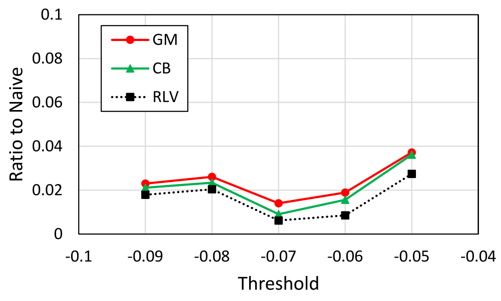

Fig. 16. PCC, communication cost (Febru).

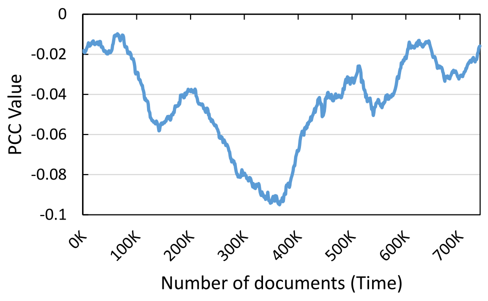

Fig. 17. PCC value over time (Febru).

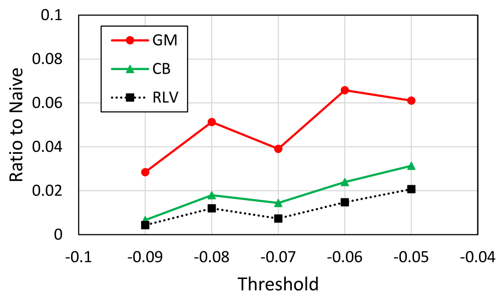

Fig. 18. PCC, communication cost (Bosnia).

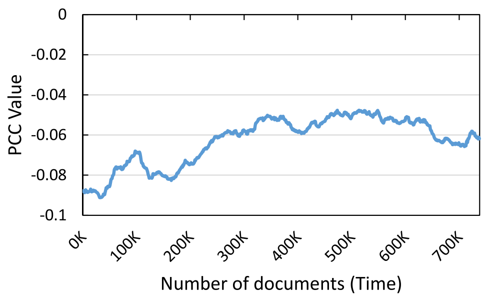

Fig. 19. PCC value over time (Bosnia).

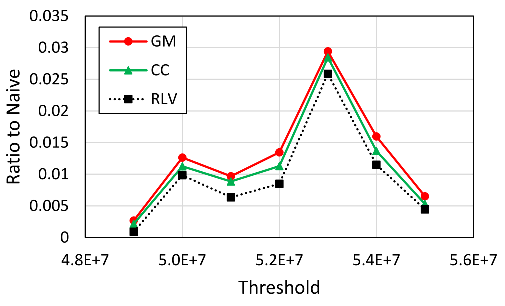

Fig. 20. Inner product, communication cost (REU).

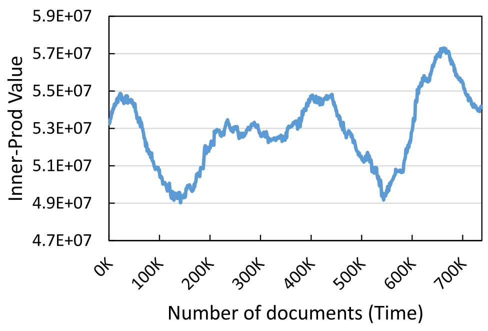

Fig. 21. Inner product value over time (REU).

To understand how the monitored threshold affects the communication overhead in Figure 16 (18), see the behavior of the function over time in Figure 17 (19). For Febru, when the threshold is equal to −0.05, it is crossed many times, rendering the monitoring task more difficult. Other values (e.g., −0.07) are less frequently crossed; hence the monitoring is more efficient. ”Bosnia” displays similar behavior, where the threshold is value of −0.06 is more frequently crossed than others.

7.2 Inner Product

CB sent fewer messages than GM for all threshold values of both datasets (see Figures 20 and 22). It is about 1.3 to 2 times better on TWIT, while only about 10–25% better on REU. Again, the proximity to the RLV graph indicates that there is little room for improvement on the REU dataset.

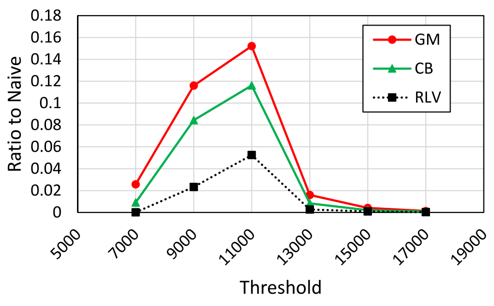

Fig. 22. Inner product, communication cost (TWIT).

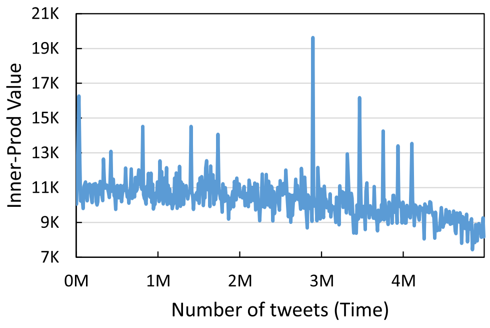

Fig. 23. Inner product value over time (TWIT).

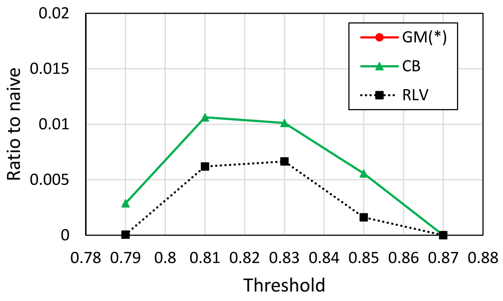

Fig. 24. Csim, communication cost (REU). *GM data is unavailable since it did not terminate in 24 hours.

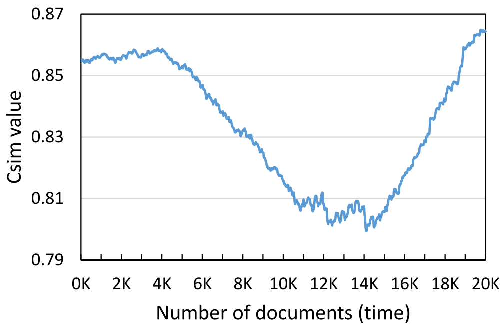

Fig. 25. Csim value over time, (REU).

To understand how the threshold affects the communication overhead in Figure 22 (20), see the behavior of the function over time in Figure 23 (21).

For the TWIT dataset, when the threshold is equal to 11,000, it is crossed many times, rendering the monitoring task more difficult. Other values (e.g., 15,000) are hardly ever crossed; hence the monitoring is more efficient. The same behavior is displayed in the REU dataset, where the threshold value of 5.3E+7 is more frequently crossed than others.

7.3 Cosine Similarity

Figures 24 and 26 show the communication cost comparison for REU and TWIT, respectively.

As noted, the GM experiments did not terminate in 24 hours. This is not entirely surprising, as monitoring Csim with GM requires solving an exceedingly difficult problem (Section 4.3.1). CB significantly improves over the naive method, reducing communication by more than two orders of magnitude for the REU dataset and by a factor of 3 to 6 for the more erratic TWIT dataset. (the runtime results are given in Table 1). The function value over time for REU and TWIT are given in Figures 25 and 27, respectively. As with the previous functions, they can be used to understand the effect of different thresholds on the communication overhead.

For the Csim function, GM runtime is so overwhelming that it can not be used in practice; still, we wanted to assess the potential communication advantage of CB over GM. To do so, instead of

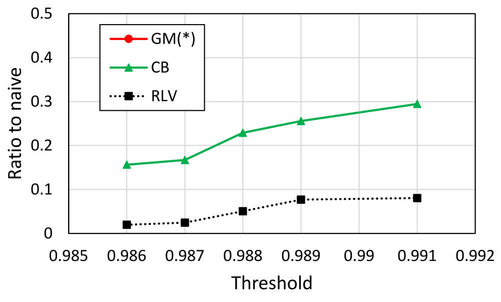

Fig. 26. Csim, communication cost (TWIT). *GM data is unavailable since it did not terminate in 24 hours.

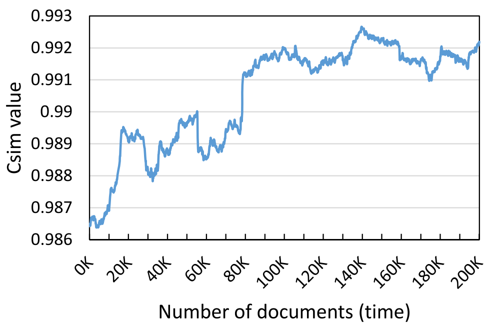

Fig. 27. Csim value over time (TWIT).

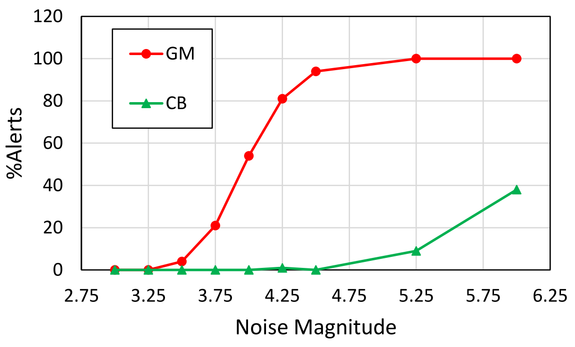

Fig. 28. Csim simulation results. Percentage of alerts per noise magnitude (lower is better). For a noise magnitude of 5.25, GM is already "saturated" with 100% percent of the points reported as violations, while CB reports less than 20% of the points.

running an experiment in a distributed setting using high-dimensional real data, we conducted an experiment using a single node on lower-dimensional simulated data.

Since the experiment is conducted on a single node, we do not report communication cost. Instead, we report the number of alerts issued by each of the methods (fewer alerts imply a better method).

Following is brief description of the experiment: A reference point $p_0 = (x_0, y_0)$ (where x_0 and y_0 are 100-dimensional vectors) was selected at random, and then a threshold T was selected such that $\text{Csim}(p_0) \leq T$. Next we selected a noise magnitude parameter, σ, and generated 1,000 vectors by adding random uniform noise in the range $[-\sigma, \sigma]$ to every component of p_0. These vectors were used as a stream of data. We repeated the experiment for different σ values.

Figure 28 summarizes the results. As expected, both methods send more alerts as the noise increases; however, CB demonstrates a clear advantage over GM.

7.4 PCA-Score

CB outperforms the other methods in terms of communication cost for all threshold values we tested. Its advantage over SN, which is the faster of the previous methods, is especially notable. Figure 29 compares the communication cost of CB to the other methods for different threshold values. As expected, all methods incur more communication for the tighter (higher) thresholds.

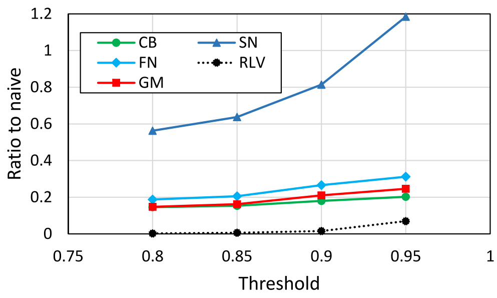

Fig. 29. PCA-Score, communication cost (KC, effective dimension 4). While the comm. cost of FN and GM is almost as good as CB's, they are two to three orders of magnitude slower (see Table 2).

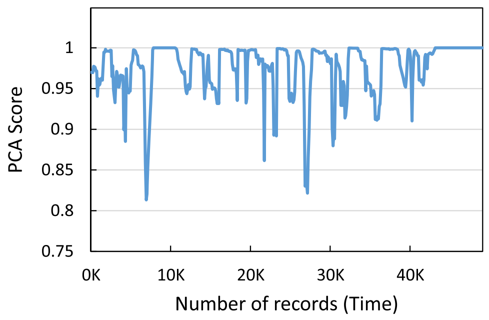

Fig. 30. PCA-Score value over time (KC).

SN's performance degrades quickly as the thresholds become tighter, while the other methods degrade more gracefully. CB's advantage is greater for the tighter thresholds (0.9 and 0.95, see Figure 30), where the monitoring task is more difficult.

8 POWER CONSUMPTION

Power consumption is becoming a critical factor; this is especially true for mobile, battery-operated devices with limited computing resources. It is expected that the computational efficiency of the CB method will be translated into superior battery lifetime. In this section, we directly evaluate the power consumption of the computational tasks for CB and GM on two resource-limited devices. Our experiments show that CB's energy consumption is orders of magnitude lower than GM's, making it much more suitable for lightweight nodes.

8.1 Preliminaries

We ran experiments on a VOYO Mini-PC and an Edison SoC, on top of the Arduino Expansion Board. The Intel Edison module is a system on chip that includes an Atom 500MHz dual-core CPU with 1GB of RAM, running Yocto Linux. Arduino is used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling physical outputs (such as lights and motors). VOYO Mini-PC is a full-fledged PC designed to be used as a smart streaming media player. It features an Intel Atom 1.33GHz quad-core CPU, with 2GB of RAM, and runs a Windows 32-bit operating system.

To evaluate the power consumption of both devices, we connected them to a stable power supply through a measuring device and measured the energy in mWh (Figure 31).

CB and GM are implemented in Python; GM, however, also requires some of Matlab's optimization packages. GM's implementation on the Mini-PC was relatively easy. However, running GM on the Edison SoC was more of a challenge, since the optimization libraries required the installation of Matlab, and Edison is memory-constrained. As a result, we were able to run only two functions on the Edison SoC: inner product (which requires no optimization) and PCC, for which we implemented a lightweight Python optimization code (using coarse grid search followed by the Powell method to find the closest point on the threshold surface).

The process for monitoring a distributed stream using CB or GM is the same except for the local condition check. In both cases the stream is parsed, the local vector is updated and checked against the local condition, and the coordinator is notified upon a violation, which it then resolves. We

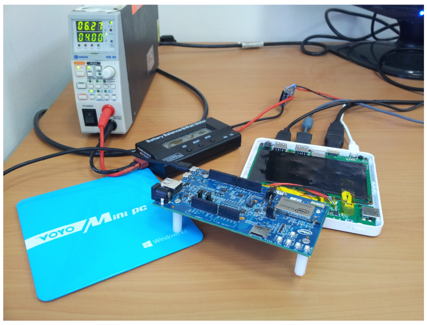
Fig. 31. The Arduino Expansion Board with Edison and VOYO Mini-PC, connected to external power measuring instruments.

wanted to evaluate the impact of the monitoring method on the power consumption of both the local condition check itself and the full monitoring process.

On each device we measured the power consumption required for two different tasks. The first task (COND) was checking the local conditions on 10,000 data items. The second task (FULL) was running the full monitoring process to digest 10,000 data items.

To improve the runtime (and power consumption) of the full GM monitoring process, we applied two effective heuristics that are described next.

8.1.1 Reducing the Running Time of GM. Recall that the local condition that GM applies consists of constructing the sphere whose diameter is the segment $\overline{p_0, p_0 + d_i}$ and checking whether it intersects with $\bar{A}$ (Section 2.1, Figure 2). This is an expensive process, which (usually) requires solving an optimization problem to find the closest point on the threshold surface.

The first heuristic arises from the observation that, if $p_0 + d_i \in \bar{A}$, there is a clear violation and no need to check for sphere intersection. Note that checking whether $p_0 + d_i \in \bar{A}$ is trivial. We can start with checking this simple condition and continue to the expensive sphere intersection only if necessary.

A second heuristic that often reduces running time was used here: first find $n(p_0)$, the point on A's boundary that is closest to p_0. Obviously, if $\|d_i\| \leq \|p_0 - n(p_0)\|$, then $p_0 + n(p_0)$ lies in C_0, the sphere whose center is p_0 and with radius $\|p_0 - n(p_0)\|$; hence, the entire sphere whose diameter is the segment $\overline{p_0, p_0 + d_i}$ lies in C_0 and does not intersect $\bar{A}$. Figure 32 schematically depicts the idea. Note that this improvement, too, requires solving the closest point problem. However, in this method, the closest point does not have to be calculated on every timestep. Furthermore, $n(p_0)$ is the same for all nodes and does not depend on the current data; therefore it can be calculated at the coordinator node (which, in some settings, is more powerful than the nodes).

These heuristics were applied only to the full monitoring process.

8.2 Results

Figure 33 summarizes the results of the power measurements on the VOYO Mini-PC. In correlation with the runtime results (Table 1), CB is orders of magnitude more power-efficient than GM for all functions except inner-prod (where CB is six times better for the COND task and two times better for the FULL task). GM's implementation for the Csim function failed to complete on real data; therefore we only present results for the COND experiment (where we used synthetic three-dimensional data). The power consumption results for the Edison SoC are depicted in Figure 34 (recall that GM could not be implemented on it for the PCA and Csim functions).

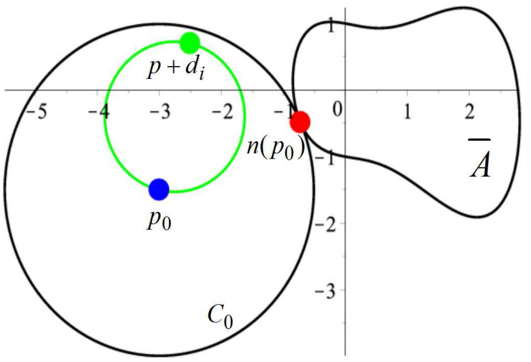

Fig. 32. Let $n(p_0)$ be the point on A's boundary closest to p_0 and C_0 the sphere whose center is at p_0 and with radius $\|p_0 - n(p_0)\|$. If $p_0 + d_i$ lies in C_0, then its entire corresponding sphere (green) lies in C_0 as well, and it is not necessary to check whether it intersects $\bar{A}$.

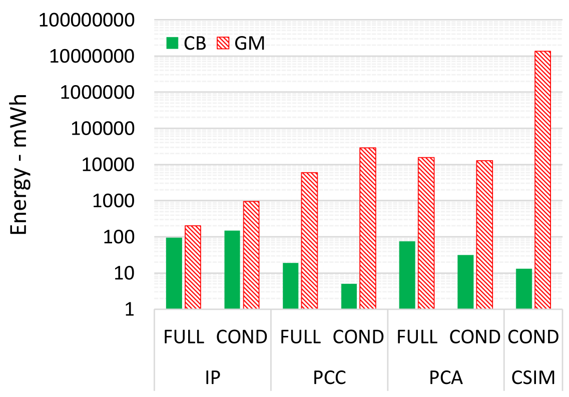

Fig. 33. VOYO Mini-Pc power consumption. GM's power consumption (striped red) is orders of magnitude higher than CB's (solid green) in most cases. *Note the logarithmic scale.*

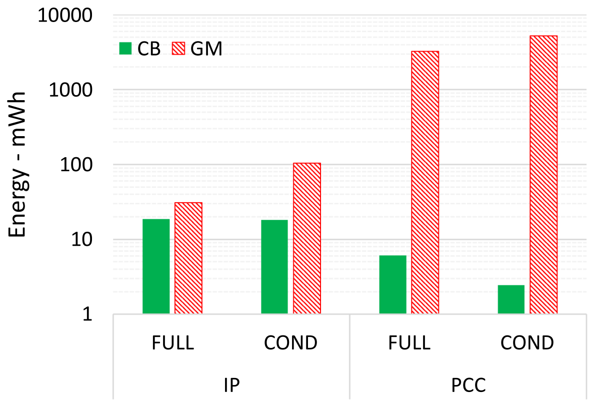

Fig. 34. Edison SoC power consumption. GM's power consumption (striped red) is higher than CB's (solid green). *Note the logarithmic scale.*

The power consumption of CB-COND is lower than that of CB-FULL for all functions (except inner-prod where the difference is negligible). This is because of the extra overhead required by the full monitoring process. On the other hand, the power consumption of GM-COND is actually higher than that of GM-FULL in all cases. This is because applying the above heuristics in the full monitoring process means that the very expansive optimization problem is sometimes skipped. We can also infer that the runtime of the monitoring process (excluding the local condition check) is negligible compared to that of the local condition check of GM. While the advantage of CB for the FULL task is smaller than its advantage for COND, it is still orders of magnitude better than GM in most cases.

9 CONCLUSIONS

Cheap, resource-constrained devices are ubiquitous, from hand-held devices and phones to sensors in cars and environmental-control systems. With the move toward the IoT, their deployment is expected to exponentially increase. Systems composed of these devices will have to perform complex monitoring tasks in real-time, thus making communication reduction a major goal. However, previous communication-efficient distributed schemes for monitoring fail on such systems due to immense computational overhead. In this article, we presented a general and efficient solution to this problem. The new method reduces orders of magnitude from the runtime overhead while keeping the communication volume to a minimum.

A APPENDIX

Some theoretical analysis, omitted from the body of the article to improve readability, is provided.

A.1 Non-Existence of an Optimal Bound in the General Case

Recall that we're given a function $f(x)$ (x is a vector) and reference point p_0, and the goal is to find an upper *convex bound* $g(x)$ that satisfies

—For all x, $g(x) \geq f(x)$.
—g satisfies some kind of optimality, i.e., is in some sense minimal among all convex bounds for f at p_0.

We denote the partial order over functions by $h_1 \succ h_2$ (where $h_1 \succ h_2$ means $h_1(x) \geq h_2(x)$ for all x).

Ideally, an *optimal* upper bound g_{opt} satisfies $g \succ g_{opt}$ for every other convex upper bound g of f. It turns out, however, that such a notion of optimality exists only when f is convex or concave (in the first case the optimal bound is f itself, and in the second case it is the tangent plane to f at p_0).

We restrict ourselves to bounds g satisfying $g(p_0) = f(p_0)$. Since $g \succ f$, this of course implies that the tangent planes of g and f are identical. The second-order Taylor expansion of g at p_0, determined by its Hessian $H_g(p_0)$, plays a crucial role (since a function is convex iff its Hessian is positive semi definite, PSD[11]). Note that higher-order terms of the Taylor expansion are dominated by the second-order ones in the vicinity of p_0, and further, the higher-order part cannot be convex or concave. Hence, it suffices to look only at the second-order Taylor expansion around p_0.

We now show that even for the simplest non-convex and non-concave function, there is no minimal element in the set of upper convex bounds.

[11]A matrix B is PSD iff $uBu^t \geq 0$ for every vector u. A symmetric matrix is PSD iff all its eigenvalues are ≥ 0.

Lemma A.1. *Let S be the set of all convex quadratics that are everywhere larger than $x^2 - y^2$. Then S has no minimal element.*

Proof. As noted, it is enough to look at upper bounds $Q(x,y)$ that satisfy $Q(0,0) = 0$ and have zero partial derivatives at $(0,0)$. We identify each such Q with a 2×2 matrix $A = [\begin{smallmatrix} a & b \\ b & c \end{smallmatrix}]$, so that $Q(x,y) = (x,y)A(x,y)^t$. The partial ordering on the quadratics corresponds to the partial ordering on matrices, where $A \geq B$ iff $A - B$ is PSD. We then need to prove that there is no minimal element among all PSD matrices greater than $[\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}]$. Let us assume such a minimal element exists, and denote it $A_0 = [\begin{smallmatrix} a_0 & b_0 \\ b_0 & c_0 \end{smallmatrix}]$. Recall that a 2×2 matrix is PSD iff $a \geq 0, ac - b^2 \geq 0$. Hence, for A_0 to be both PSD and for $A_0 \geq [\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}]$ to hold, we must have $a_0c_0 - b_0^2 \geq 0, (a_0 - 1)(c_0 + 1) - b_0^2 \geq 0$. If these two inequalities are strict, then it follows from a trivial continuity consideration that A_0 can be made smaller by subtracting $[\begin{smallmatrix} \epsilon & 0 \\ 0 & \epsilon \end{smallmatrix}]$ from it for a small enough ϵ, such that the resulting matrix will still be both PSD and $\geq [\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}]$, hence contradicting minimality. Assume next that $a_0c_0 - b_0^2 = 0, (a_0 - 1)(c_0 + 1) - b_0^2 > 0$. Now, we can perturb the elements of A_0 to obtain A_0' by $a_0' = a_0 + \epsilon, c_0' = c_0 - \frac{c_0\epsilon}{a_0+\epsilon}, b_0' = b_0$, where ϵ is chosen to be positive and small enough so that $(a_0' - 1)(c_0' + 1) - b_0'^2 > 0$ and $c_0 - \frac{c_0\epsilon}{a_0+\epsilon} \geq 0$ (again, such an ϵ exists due to continuity considerations). The perturbation is chosen such that $a_0'c_0' - b_0'^2 = a_0c_0 - b_0^2 = 0$. Now, $A_0' - A_0 = E_0$, for $E_0 = [\begin{smallmatrix} \epsilon & 0 \\ 0 & -\frac{c_0\epsilon}{a_0+\epsilon} \end{smallmatrix}]$; hence E_0 is clearly not PSD, so $A_0' \in S$ and $A_0' \ngeq A_0$, again contradicting the minimality of A_0. Similar considerations hold for the case in which $a_0c_0 - b_0^2 > 0, (a_0 - 1)(c_0 + 1) - b_0^2 = 0$. We can therefore assume that $a_0c_0 - b_0^2 = 0, (a_0 - 1)(c_0 + 1) - b_0^2 = 0$. It then follows that $c_0 = a_0 - 1$, and hence $A_0 = [\begin{smallmatrix} a_0 & \sqrt{a_0(a_0-1)} \\ \sqrt{a_0(a_0-1)} & a_0 - 1 \end{smallmatrix}]$. We complete the proof by showing that the set of such matrices is totally unordered – that is, if $A_0' = [\begin{smallmatrix} a_0' & \sqrt{a_0'(a_0'-1)} \\ \sqrt{a_0'(a_0'-1)} & a_0' - 1 \end{smallmatrix}]$, then $A_0 \ngeq A_0', A_0' \ngeq A_0$. To see this, assume W.L.O.G that $a_0 > a_0'$. Let us look at $A_0 - A_0' = [\begin{smallmatrix} a_0 - a_0' & \sqrt{a_0(a_0-1)} - \sqrt{a_0'(a_0'-1)} \\ \sqrt{a_0(a_0-1)} - \sqrt{a_0'(a_0'-1)} & a_0 - a_0' \end{smallmatrix}]$. The leading diagonal entry is positive, but the determinant is strictly negative (proving this is just a rudimentary exercise). Hence, $A_0 - A_0'$ is not PSD nor NSD (negative semi definite), so neither $A_0 \geq A_0'$ nor $A_0' \geq A_0$ holds. This concludes the proof. □

We conclude by noting that the proof immediately extends to *any* quadratic in n variables $x_1 \ldots x_n$, which is non-convex and non-concave, since the matrix defining it must contain at least one positive and at least one negative eigenvalue. Hence up to rotation and scale it can be expressed as $x_1^2 - x_2^2 \pm x_2^3 \ldots \pm x_2^n$, and the proof proceeds by applying the above lemma to the $x_1^2 - x_2^2$ part.

We proved that there is no minimal convex bound among quadratics. we continue to prove the general case:

Lemma A.2. *Let G be the set of all convex functions that are everywhere larger than $x^2 - y^2$. Then G has no minimal element.*

Proof. Recall that all convex quadratics that are everywhere larger than $f(x,y) = x^2 - y^2$ take the form $Q(x,y) = (x,y)A(x,y)^t$, where $A = [\begin{smallmatrix} a & \sqrt{a(a-1)} \\ \sqrt{a(a-1)} & a - 1 \end{smallmatrix}]$ and $a \geq 1$, or use function notation $Q(x,y) = ax^2 + 2\sqrt{a(a-1)}xy + (a-1)y^2$.

Denote by $Q_a(x,y)$ the quadratic bound $Q(x,y)$, for a specific choice of a; for example, $Q_1(x,y) = x^2$ and $Q_2(x,y) = 2x^2 + 2\sqrt{2}xy + y^2$.

Assume that there exists a function $g(x, y)$ that is a minimal convex bound of $f(x, y)$, as follows:

(1) $g(x, y) \succ f(x, y)$.
(2) g is convex.
(3) $g(x, y) \prec Q_a(x, y)$ for $a \geq 1$.

To guarantee minimality, $g(x, y)$ must not be larger than $Q_a(x, y)$, in particular $g(x, y) \prec Q_1(x, y)$ and $g(x, y) \prec Q_2(x, y)$. Let $p_0 = (1, -\sqrt{2})$, and $p_1 = (0, \sqrt{2})$. We note that $Q_2(p_0) = 0$ and $Q_1(p_1) = 0$. Therefore $g(p0) \leq 0$ and $g(p1) \leq 0$. Since $g(x, y)$ is convex, $g(\frac{p_0+p_1}{2}) \leq \frac{g(p_0)+g(p_1)}{2} \leq 0$. However, $f(\frac{p_0+p_1}{2}) = f(0.5, 0) = 0.25$, so $g(0.5, 0) \leq 0 < f(0.5, 0)$ in contradiction to $g(x, y)$ being an upper bound of $f(x, y)$. □

A.2 Applying GM to PCA-Score Monitoring

To monitor the PCA-Score in the GM framework, it is necessary to check whether a sphere in matrix space is contained in the admissible region A. Here, A consists of all matrices M whose eigenvalues satisfy the inequality in Equation (5). Hence, to check whether a sphere lies in the admissible region, we must check that the eigenvalues of every matrix in it satisfy $(\sum_{1 \leq i \leq k} \lambda_i^2)/(\sum_{1 \leq i \leq m} \lambda_i^2) \geq T$. To solve this problem, we must relate the change in the eigenvalues to the change in the matrix elements. This can be done using perturbative bounds on the eigenvalues; for a review on such bounds, see, e.g., Chapter 8.1.2 in Reference [25]. Such bounds are also used in References [28, 29], which studied distributed PCA monitoring for system health analysis. The results are summarized below:

Lemma A.3. *For two symmetric $n \times n$ matrices A, B, the following inequality holds:* $\sum_{i=1}^{n} [\lambda_i(A+B) - \lambda_i(A)]^2 \leq \|B\|_F^2$*, where $\|B\|_F$ is the* FN, *defined as* $\sqrt{\sum_{i,j} B_{i,j}^2}$.
This celebrated result is known as the Wielandt-Hoffman Theorem.

Lemma A.4. *Using the same notation as in Lemma A.3, the following inequality holds for every $1 \leq k \leq n$: $|\lambda_i(A+B) - \lambda_i(A)| \leq \|B\|_2$, where $\|B\|_2$ is B's* SN *(which, for symmetric matrices, equals $\max\{|\lambda_1(B)|, |\lambda_n(B)|\}$).*

We refer to the methods that employ the bounds in Lemma A.3 (resp. Lemma A.4) according to the type of perturbative bounds they apply, i.e., Frobenius (resp. spectral) Norm.

A.3 Proof of Concavity for Pearson Correlation Monitoring

In Section 4.1, we used the fact that one of the components of the Pearson correlation function, $\sqrt{x - x^2}\sqrt{y - y^2}$, is concave. The proof follows. We use two well-known facts:

— A function is concave iff its Hessian is negative semidefinite.
— A 2×2 symmetric matrix $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix}$ is negative semidefinite iff $a_{11} \leq 0, |A| = a_{11}a_{22} - a_{12}^2 \geq 0$.

Directly calculating the Hessian yields

$$\frac{1}{4}\begin{bmatrix} \frac{\sqrt{y(1-y)}}{x(x-1)\sqrt{x(1-x)}} & \frac{(2x-1)(2y-1)}{\sqrt{x(1-x)}\sqrt{y(1-y)}} \\ \frac{(2x-1)(2y-1)}{\sqrt{x(1-x)}\sqrt{y(1-y)}} & \frac{\sqrt{x(1-x)}}{y(y-1)\sqrt{y(1-y)}} \end{bmatrix}.$$

Clearly, $a_{11} \leq 0$ due to the $x - 1$ factor in the denominator (recall that we're only interested in the range $0 \leq x, y \leq 1$). The determinant of the Hessian equals

$$\frac{x(1-x) + y(1-y) - 4xy(1-x)(1-y)}{4x\,(1-x)\,y\,(1-y)} \ .$$

The denominator is obviously positive. To see that the numerator is positive, first expand it to get $-4\,x^2y^2 + 4\,x^2y + 4\,xy^2 - x^2 - 4\,xy - y^2 + x + y$, next modify it by replacing $x + y$ with $x^2 + y^2$ (since $0 \leq x, y \leq 1$, this obviously decreases the overall value). Then, it is trivial to see that the new expression equals $4xy(1-x)(1-y)$, which is obviously positive; hence the original expression is positive as well.

A.4 Proof for Eigenvalues for Cosine Similarity

To apply the convexity gauge for the cosine similarity function, we need to compute the eigenvalues of the function $\|x\|\ \|y\|$. We note that this function is *rotationally symmetric* (under rotations of x and y); this follows from the fact that rotation preserves norms. Hence the Hessian's eigenvalues are invariant to rotations in x and y. Obviously, we can rotate any vector u to the vector $(\|u\|, 0, 0\ldots0)$; thus it suffices to compute the eigenvalues of the Hessian at the point $p \triangleq [(\|x\|, 0, 0\ldots0), (\|u\|, 0, 0\ldots0)]$. At this point the Hessian assumes a very simple form, shown below for $n = 4$ with the obvious generalization to any dimension:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{\|y\|}{\|x\|} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\|y\|}{\|x\|} & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\|x\|}{\|y\|} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\|x\|}{\|y\|} \end{bmatrix} .$$

It is easy to verify that the eigenvalues are 1,−1, and $\frac{\|y\|}{\|x\|}, \frac{\|x\|}{\|y\|}$, each with multiplicity $n - 1$.

A.5 Computing the Closest Point for Cosine Similarity Surface

To apply GM to cosine similarity monitoring, we must be able to find the closest point on the threshold surface $\langle x, y\rangle = T\|x\|\ \|y\|$. The corresponding optimization problem is

$$\begin{aligned} &\text{Minimize } \frac{1}{2}(\|x - x_0\|^2 + \|y - y_0\|^2) \\ &\text{such that } \langle x, y\rangle - T\|x\|\ \|y\| = 0 \ . \end{aligned} \tag{7}$$

We first write $x = \alpha u, y = \beta v$, where α, β are scalars and u, v unit vectors. It is easier to work in this representation, since then the condition $\langle x, y\rangle - T\|x\|\ \|y\| =$ can be written as $\langle u, v\rangle - T = 0$. Putting it all together yields the problem

$$\begin{aligned} &\text{Minimize } \frac{1}{2}(\|\alpha u - x_0\|^2 + \|\beta v - y_0\|^2) \\ &\text{such that } \langle u, v\rangle - T = 0, \|u\|^2 - 1 = 0, \|v\|^2 - 1 = 0. \end{aligned}$$

Next, we introduce three Lagrange multipliers for the three constraints:

$$\begin{aligned} F \triangleq\ & \frac{1}{2}(\|\alpha u - x_0\|^2 + \|\beta v - y_0\|^2) \\ & + \lambda_1(\langle u, v\rangle - T) + \frac{1}{2}\lambda_2(\|u\|^2 - 1) + \frac{1}{2}\lambda_3(\|v\|^2 - 1). \end{aligned}$$

Taking the derivative of F by α yields $(\langle u, \alpha u - x_0 \rangle = 0$; hence $\alpha = \langle u, x_0 \rangle$. Similarly, $\beta = \langle v, y_0 \rangle$. After some simple manipulations, the problem can be written as

$$\begin{aligned} &-\frac{1}{2}(\langle x_0, u\rangle^2 + \langle y_0, v\rangle^2) \\ &\quad + \lambda_1(\langle u, v\rangle - T) + \frac{1}{2}\lambda_2(\|u\|^2 - 1) + \frac{1}{2}\lambda_3(\|v\|^2 - 1). \end{aligned}$$

Taking the derivative by u yields

$$-\langle x_0, u\rangle x_0 + \lambda_1 u + \lambda_3 v = 0. \tag{8}$$

Now take the inner product with u to obtain

$$-\langle x_0, u\rangle^2 + \lambda_1 + \lambda_3 T = 0 \Rightarrow \langle x_0, u\rangle = \sqrt{\lambda_1 + \lambda_3 T}.$$

Substituting back in Equation (8), and repeating the process for $\langle y_0, u\rangle$, yields the pair of equations

$$-\sqrt{\lambda_1 + \lambda_3 T} + \lambda_1 u + \lambda_3 v = 0, -\sqrt{\lambda_2 + \lambda_3 T} + \lambda_2 v + \lambda_3 u = 0.$$

These equations can be solved to write u, v as functions of $x_0, y_0, \lambda_1, \lambda_2, \lambda_3$. Then, finally, the conditions $\|u\|^2 = 1, \|v\|^2 = 1\langle u, v\rangle = T$ yields three equations in the unknowns $\lambda_1, \lambda_2, \lambda_3$.

While the form of the resulting equations is independent of the dimension of the vectors, their solution turns out to be exceedingly difficult. We have tried applying the GloptiPoly package [27], which is dedicated to finding all roots of a set of algebraic equations, but it could not find a solution. The Matlab package took on the average 3 minutes to find a solution, but sometimes it missed part of the solutions. The most successful in solving the equations was Maple$^{\text{TM}}$, but its symbolic package could not even solve the case where x, y are of dimension 2; its numerical "fsolve" function was able to find all solutions, but the average running time, too, was about 3 minutes.

ACKNOWLEDGMENTS

We would like to thank Dr. Avi Yadgar for his help with the power consumption experiments.

REFERENCES

[1] Amir Ali Ahmadi and Georgina Hall. 2015. DC decomposition of nonconvex polynomials with algebraic techniques. *Mathematical Programming* (2015), 1–26.

[2] Chrisil Arackaparambil, Joshua Brody, and Amit Chakrabarti. 2009. Functional monitoring without monotonicity. In *ICALP*. 95–106.

[3] B. Babcock and C. Olston. 2003. Distributed top-k monitoring. In *SIGMOD*. 28–39.

[4] Shivnath Babu and Jennifer Widom. 2001. Continuous queries over data streams. *SIGMOD* 30, 3 (2001), 109–120.

[5] Marco Balduini, Irene Celino, Daniele Dell'Aglio, Emanuele Della Valle, Yi Huang, Tony Lee, Seon-Ho Kim, and Volker Tresp. 2012. BOTTARI: An augmented reality mobile application to deliver personalized and location-based recommendations by continuous analysis of social media streams. *Web Semant.* 16 (2012), 33–41.

[6] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. 2002. Counting distinct elements in a data stream. In *RANDOM*. 1–10.

[7] Steven Bird. 2006. NLTK: The natural language toolkit. In *COLING/ACL*. 69–72.

[8] S. Boyd and L. Vandenberghe. 2004. *Convex Optimization.* Cambridge University Press.

[9] Joshua Brody and Amit Chakrabarti. 2009. A multi-round communication lower bound for gap hamming and some consequences. In *CCC*. 358–368.

[10] Sabbas Burdakis and Antonios Deligiannakis. 2012. Detecting outliers in sensor networks using the geometric approach. In *ICDE*. 1108–1119.

[11] Graham Cormode. 2013. The continuous distributed monitoring model. *SIGMOD Rec.* 42, 1 (2013), 5–14.

[12] Graham Cormode and Minos N. Garofalakis. 2005. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*. 13–24.

[13] Graham Cormode and Minos N. Garofalakis. 2008. Approximate continuous querying over distributed streams. *ACM Trans. Database Syst.* 33, 2 (2008), 1–39.

[14] Abhinandan Das, Sumit Ganguly, Minos N. Garofalakis, and Rajeev Rastogi. 2004. Distributed set expression cardinality estimation. In *VLDB*. 312–323.

[15] Mark Dilman and Danny Raz. 2002. Efficient reactive monitoring. *SAC* 20, 4 (2002), 668–676.

[16] Manfredo P. Do Carmo. 2016. *Differential Geometry of Curves and Surfaces: Revised and Updated* (2nd ed.). Courier Dover Publications.

[17] Ky Fan. 1949. On a theorem of weyl concerning eigenvalues of linear transformations I. In *Proc. Natl. Acad. Sci. U.S.A* 35, 11 (1949), 652–655.

[18] Arik Friedman, Izchak Sharfman, Daniel Keren, and Assaf Schuster. 2014. Privacy-preserving distributed stream monitoring. In *NDSS*. 1–12.

[19] Moshe Gabel, Daniel Keren, and Assaf Schuster. 2015. Monitoring least squares models of distributed streams. In *SIGKDD*. ACM, 319–328.

[20] Moshe Gabel, Daniel Keren, and Assaf Schuster. 2017. Anarchists, unite: Practical entropy approximation for distributed streams. KDD, 837–846.

[21] Moshe Gabel, Assaf Schuster, and Daniel Keren. 2014. Communication-efficient distributed variance monitoring and outlier detection for multivariate time series. In *IPDPS*. 37–47.

[22] Minos N. Garofalakis, Daniel Keren, and Vasilis Samoladas. 2013. Sketch-based geometric monitoring of distributed stream queries. *PVLDB* 6, 10 (2013), 937–948.

[23] Nikos Giatrakos, Antonios Deligiannakis, Minos Garofalakis, Izchak Sharfman, and Assaf Schuster. 2014. Distributed geometric query monitoring using prediction models. *TODS* 39, 2 (2014), 16:1–16:42.

[24] Nikos Giatrakos, Antonios Deligiannakis, Minos N. Garofalakis, Izchak Sharfman, and Assaf Schuster. 2012. Prediction-based geometric monitoring over distributed data streams. In *SIGMOD*. 265–276.

[25] G. H. Golub and C. F. Van Loan. 1996. *Matrix Computations,* (3rd ed.). Johns Hopkins University Press.

[26] Rajeev Gupta, Krithi Ramamritham, and Mukesh K. Mohania. 2013. Ratio threshold queries over distributed data sources. In *Proceedings of the VLDB Endowment* 6, 8 (2013), 565–576.

[27] Didier Henrion, Jean-Bernard Lasserre, and Johan Löfberg. 2009. GloptiPoly 3: Moments, optimization and semidefinite programming. 24, 4–5 (2009), 761–779.

[28] Ling Huang, Michael I. Jordan, Anthony Joseph, Minos Garofalakis, and Nina Taft. 2006. In-network PCA and anomaly detection. In *In NIPS*. 617–624.

[29] 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*. 134–142.

[30] Antonios Igglezakis, Antonios Deligiannakis, and Aggelos Bletsas. 2014. Geometric monitoring for CSI reduction in amplify-and-forward relay networks. In *ICASSP*. 2729–2733.

[31] S. M. Riazul Islam, Daehan Kwak, M. D. Humaun Kabir, Mahmud Hossain, and Kyung-Sup Kwak. 2015. The internet of things for health care: A comprehensive survey. *IEEE Access* 3 (2015), 678–708.

[32] S. Ratnasamy Jain, J. M. Hellerstein, and D. Wetherall. 2004. A wakeup call for internet monitoring systems: The case for distributed triggers. In *HotNets-III*. 1–6.

[33] Jiong Jin, Jayavardhana Gubbi, Slaven Marusic, and Marimuthu Palaniswami. 2014. An information framework for creating a smart city through internet of things. *IEEE Internet Things J.* 1, 2 (2014), 112–121.

[34] Bhargav Kanagal and Amol Deshpande. 2008. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE*. 1160–1169.

[35] Srinivas R. Kashyap, Jeyashankher Ramamirtham, Rajeev Rastogi, and Pushpraj Shukla. 2008. Efficient constraint monitoring using adaptive thresholds. In *ICDE*. 526–535.

[36] Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. 2006. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD*. 289–300.

[37] Daniel Keren, Guy Sagy, Amir Abboud, David Ben-David, Assaf Schuster, Izchak Sharfman, and Antonios Deligiannakis. 2014. Geometric monitoring of heterogeneous streams. *TKDE* 26, 8 (2014), 1890–1903.

[38] Daniel Keren, Izchak Sharfman, Assaf Schuster, and Avishay Livne. 2012. Shape sensitive geometric monitoring. *TKDE* 24, 8 (2012), 1520–1535.

[39] Anukool Lakhina, Mark Crovella, and Christophe Diot. 2004. Diagnosing network-wide traffic anomalies. In *SIGCOMM*. 219–230.

[40] 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 *DEBS*. 203–214.

[41] Arnon Lazerson, Daniel Keren, and Assaf Schuster. 2016. Lightweight monitoring of distributed streams. In *KDD*. 1685–1694.

[42] Arnon Lazerson, Izchak Sharfman, Daniel Keren, Assaf Schuster, Minos N. Garofalakis, and Vasilis Samoladas. 2015. Monitoring distributed streams using convex decompositions. *PVLDB* 8, 5 (2015), 545–556.

[43] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.* 5 (2004), 361–397.
[44] Feifei Li, Ke Yi, and Jeffrey Jestes. 2009. Ranking distributed probabilistic data. In *SIGMOD*. 361–374.
[45] 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*. 1023–1031.
[46] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. 2005. TinyDB: An acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.* 30, 1 (2005), 122–173.
[47] Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. 2005. KLEE: A framework for distributed top-k query algorithms. In *Proceedings of the VLDB Endowment*. 637–648.
[48] Ilya S. Molchanov and Pedro Terán. 2003. Distance transforms for real-valued functions. *J. Math. Anal. Appl.* 278, 2 (2003), 472–484.
[49] Oluwole Okunola, A. Uzairu, C. Gimba, and G. Ndukwe. 2012. Assessment of gaseous pollutants along high traffic roads in Kano, Nigeria. *Intl. J. Environment Sustainability* 1, 1 (2012).
[50] Themis Palpanas. 2013. Real-time data analytics in sensor networks. In *Managing and Mining Sensor Data*. 173–210.
[51] Themistoklis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. 2003. Distributed deviation detection in sensor networks. *SIGMOD Record* 32, 4 (2003), 77–82.
[52] Odysseas Papapetrou and Minos Garofalakis. 2014. Continuous fragmented skylines over distributed streams. In *ICDE*. 124–135.
[53] Jeff M. Phillips, Elad Verbin, and Qin Zhang. 2012. Lower bounds for number-in-hand multiparty communication complexity, made easy. In *SODA*. 486–501.
[54] Mohammad Rouhani and Angel Domingo Sappa. 2012. Implicit polynomial representation through a fast fitting error estimation. *IEEE T. Image. Process.* 21, 4 (2012), 2089–2098.
[55] Guy Sagy, Daniel Keren, Izchak Sharfman, and Assaf Schuster. 2010. Distributed threshold querying of general functions by a difference of monotonic representation. In *Proceedings of the VLDB Endowment* 4, 2 (2010), 46–57.
[56] Shetal Shah and Krithi Ramamritham. 2008. Handling non-linear polynomial queries over dynamic data. In *ICDE*. 1043–1052.
[57] IzchaK. Sharfman, Assaf Schuster, and Daniel Keren. 2006. A geometric approach to monitoring threshold functions over distributed data streams. In *SIGMOD*. 301–312.
[58] IzchaK. Sharfman, Assaf Schuster, and Daniel Keren. 2007. Aggregate threshold queries in sensor networks. In *IPDPS*. 1–10.
[59] IzchaK. Sharfman, Assaf Schuster, and Daniel Keren. 2007. A geometric approach to monitoring threshold functions over distributed data streams. *ACM Trans. Database Syst.* 32, 4 (2007), 23.
[60] IzchaK. Sharfman, Assaf Schuster, and Daniel Keren. 2008. Shape sensitive geometric monitoring. In *PODS*. 301–310.
[61] Marshall H. Stone. 1948. The generalized weierstrass approximation theorem. *Math. Mag.* 21, 5 (1948), 237–254.
[62] Mingwang Tang, Feifei Li, Jeff M. Phillips, and Jeffrey Jestes. 2012. Efficient threshold monitoring for distributed probabilistic data. In *ICDE*. 1120–1131.
[63] Ran Wolff. 2015. Distributed convex thresholding. In *PODC*. 325–334.
[64] Ran Wolff, Kanishka Bhaduri, and Hillol Kargupta. 2009. A generic local algorithm for mining data streams in large distributed systems. *TKDE* 21, 4 (2009), 465–478.
[65] James Yeh. 2006. *Real Analysis: Theory of Measure and Integration Second Edition*. World Scientific Publishing Company.
[66] Gal Yehuda, Daniel Keren, and Islam Akaria. 2017. Monitoring properties of large, distributed, dynamic graphs. In *IPDPS*. 2–11.
[67] B.-K. Yi, Nikolaos D. Sidiropoulos, Theodore Johnson, H. V. Jagadish, Christos Faloutsos, and Alexandros Biliris. 2000. Online data mining for co-evolving time sequences. In *ICDE*. 13–22.
[68] Yunyue Zhu and Dennis Shasha. 2002. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*. 358–369.

Received September 2017; revised March 2018; accepted May 2018