Guaranteed Approximation and Bandwidth Efficient Distributed Function Monitoring Schemes

Yuval Alfassi
Computer Science Department
University of Haifa
Haifa, Israel
yuvalalfassi@gmail.com

Dani Keren
Computer Science Department
University of Haifa
Haifa, Israel
dkeren@cs.haifa.ac.il

Moshe Gabel
Computer Science Faculty
Technion
Haifa, Israel
mgabel@cs.technion.ac.il

Assaf Shuster
Computer Science Faculty
Technion
Haifa, Israel
assaf@cs.technion.ac.il

Gal Yehuda
Computer Science Faculty
Technion
Haifa, Israel
gal2016@gmail.com

***Abstract*— Distributed monitoring is a problem that arises when trying to monitor properties of dynamic data which is spread distributively. Tracking the value of a function over dynamic data in a distributed setting is a challenging problem in numerous real-world modern applications. Several monitoring schemes were proposed as an approach to coping with this problem in order to reduce the amount of communication messages between servers, as well as the communication bandwidth.**

Here, we propose several new distributed monitoring schemes using much less communication bandwidth. Existing schemes send high dimensional vectors from server to server while we propose some innovative methods for reducing the dimensionality of the transmitted data even down to one single scalar.

One scheme we propose is the *Value Scheme* which exploits some traits of convex functions, and another is the *Distance Scheme* which treats the function monitoring problem as a geometric monitoring problem, thereby utilizing geometric distances for distributed monitoring.

Moreover, we present a clever way to incorporate lossy data-sketches into the schemes, which influence the bandwidth as well.

I. Introduction

Monitoring a function over large amount of dynamically changed data in a distributed fashion is a common computer-science challenge. Whether it's monitoring features of distributed sensor networks [**?**], top-k monitoring [**?**], monitoring distributed ratio queries [**?**] or tracking properties of large distributed dynamic graphs [**?**], innovative approaches had to be developed in order to deal with the difficulties of both the data being dynamic and distributed.

The need of minimizing both the bandwidth and the processing power is expressed in [**?**]; a good example is internet of things objects which are operated on batteries, hence sending data via a communication channel should be minimized [*]. Furthermore, in the *Big Data* era, where data is of very high dimensionality and changes rapidly, data transmition over a communication channel has to be planned cleverly. Transmition of high dimensional data is not only impractical, but also extremely time consuming; for instance, air pollution sensors which distributively have to determine the air pollution level may benefit from an economical communication approach [**?**].

Previous works were made on tracking the distributed value of linear functions [**?**], where the linear properties make the distributed monitoring much easier; though, in order to monitor non-linear functions, a handful of difficulties arise. Works were done on distributively monitoring the entropy of distributed streams [**?**] [**?**], monitoring the inner-product value of distributed dynamic vectors [**?**], monitoring distributevely least squares models [**?**], and monitoring the number of triangles of dynamic graphs held in distributed servers [**?**].

It should be noted that the classic approach to distributed monitoring is the "periodic polling" technique [**?**] where the central server – *the coordinator*, polls the distributed data-servers for their observation at a certain frequency. This method may miss peaks of critical global data-changes, and also may be more expensive communication-wise – high dimensional data is transmitted even when the changes are small.

A. Problem Definition

Commonly, distributed monitoring schemes are focused on determining whether a function over dynamic distributed data crosses a certain threshold. This is used as a component to the *distributed function approximation problem* [**?**], which ε-approximates the value of a function over time.

The distributed model is described as follows:

1) There are n data-servers, $server_1 ... server_n$

2) A central *coordinator* exists, with whom the servers communicate.
3) $Server_i$ knows only its dynamic *local vector* v_i.
4) The *global vector* v is the average of the local vectors:

$$v = \frac{1}{n}\sum_{i=1}^{n} v_i \tag{1}$$

5) A function f is monitored over the *global vector* v so it's ε-approximated with 100% confidence: let the estimation be the dynamic value μ (without loss of generality, assume $\mu \geq 0$), then at all times:

$$(1-\varepsilon)\mu \leq f(v) \leq (1+\varepsilon)\mu \tag{2}$$

The *threshold monitoring problem* [**?**] monitors whether the function's value crosses a certain threshold. The function approximation problem is commonly reduces to two simultaneous threshold monitoring problems: let $T = (1+\varepsilon)\mu$ be the upper-bound threshold's value, then, the upper-bound monitoring objective is to determine whether:

$$f(v) \leq T \tag{3}$$

Likewise, the lower-bound function monitoring is done with the threshold $(1-\varepsilon)\mu$.
In turn, this threshold monitoring can be treated as a *geometric monitoring problem* [**?**], where one tries to find a *safe zone* of vectors $\{v \mid f(v) \leq T\}$, which is a convex, so every linear combination of vectors inside this *safe zone* is also inside the *safe zone*, see Fig. **??**. This geometric *safe zone* approach is the fundamental idea behind previous distributed monitoring techniques.

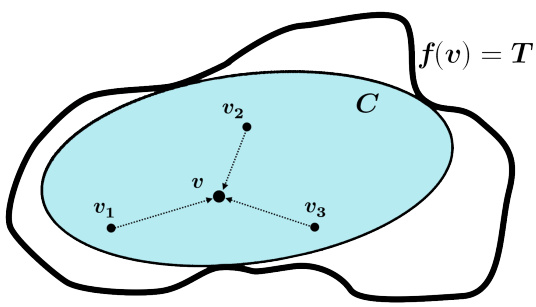

Fig. 1: Convex Safe Zone

C is a convex subspace of $\{v \mid f(v) \leq T\}$ so since $v_1, v_2, v_3 \in C$, so does the average vector $v \in C$.

B. Distributed Monitoring Initiation

When initiating the protocol of the distributed monitoring scheme, its often assumed the initial global vector v is known to *the coordinator* as well to the local servers. In order to eliminate bias toward a certain server and to prevent local violations initially [**?**], the local vectors refer to the initial global vector v as their *reference vector*, noted by r_0: the monitoring is done so the servers doesn't operate on their local data vector but the reference vector plus their local dynamic data-change. Since at all times the global vector is still the average of the local vectors, this step is permitted. Note the change vectors as $ch_1 ... ch_n$, then:

$$v = \frac{v_1 + ... + v_n}{n} = \frac{(r_0 + ch_1) + ... + (r_0 + ch_n)}{n} \tag{4}$$

So the local vector $server_i$ maintains the vector $v_i = r_0 + ch_i$ instead of the real data-vector.
A *monitoring violation* occurs when a local server suspects that due to the current local change vector the function approximation or the threshold condition may not hold anymore – so either this suspicion is true, and a *true violation* occurs, so a *Full Sync* has to be invoked, or this was a *false alarm* and a violation resolution protocol has to be done according to the distributed monitoring scheme.
The *Full Sync* gathers the whole data changes from the servers, recomputes the global average vector v, and transmits it to the servers so they set it to be their new reference point, while zeroing out the change vectors. In addition, the function's approximation is reset to the current value of $f(v)$ (since v is known), and the lower-bound and upper-bound thresholds of the monitoring are set to $(1 \pm \varepsilon)f(v)$.
The communication channel is used when a local violation takes place; bandwidth is specifically wasted when *false alarms* occur. Consequently, the distributed monitoring schemes we propose decrease the dimensionality of the transmitted data when a *false alarm* takes place, in regard to the quantity of the occurrences of the *false alarms* of the protocol.

C. Contributions

1) Introducing multiple innovative distributed monitoring schemes which avoid sending big dimensional data unless its crucially needed.
2) Proving the *Distance Lemma*, a lemma used as a basis of a distributed monitoring scheme we present. The *Distance Lemma* states that given a convex body and several points, if the sum of distances to the convex border from the points inside the convex body is greater than the sum of distances to the border from the points on the outside, then the average of the points is inside the convex body.
3) Incorporating data-sketches into distributed monitoring schemes without damaging the 0% false-negative necessity of the distributed monitoring problem; i.e., we managed to prevent having the probabilistic nature of the sketches affect the distributed monitoring having a false-positive result.
4) Conducting several experiments, laying out comparisons of multiple attributes of distributed monitoring schemes on real-world data, focusing on the bandwidth consumption.

II. Previous Work

A. Linear Functions

Since linear functions are additive and homogeneous, the basic algorithm for distributively monitoring their value is fairly easy. Since $f(v) = \frac{1}{n}\sum f(v_i)$, tracking the value of the global $f(v)$ isn't quite complicated. A work about linear functions such as the distributed count problem was done at [**?**]. However, things get more sophisticated when dealing with non-linear functions.

B. The Covering Spheres Method

The first approach which exploited some geometric traits of the ditributed monitoring problem is the *Covering Spheres* method [**?**]. This method is based on constraints on the local change-vectors.

Let r_0 be the reference point of the servers and ch_i the change vector of $server_i$. Then, $server_i$ stays quiet if the sphere whose diameter is the segment between r_0 and ch_i is fully inside the space $\{v \mid f(v) \leq T\}$. Turns out, this method artificially creates a convex safe zone subspace (as shown in Fig. **??**) where change vectors could be at without a need for communication.

This method seemed very effective theoratically, though it proved to be impractical. The *Covering Spheres* method demands performing lots of time consuming heavy mathematical operations, so it isn't scalable computation-wise; moreover, this method requires computing distances to a high dimensional complicated surface, which does not have a closed mathematical solution nor a good approximation [**?**].

Furthermore, the violation resolution phase demands transmitting high dimensional vectors, which makes the communication bandwidth quite too high.

C. The Convex Decomposition Method

Another distributed monitoring scheme previously developed is the *Convex Decomposition* method [**?**]. This method also treats the function monitoring problem as a geometric monitoring problem. The *Convex Decomposition* composes a convex *safe zone* by decomposing the space $\{v \mid f(v) \leq T\}$ into convex subspaces and geometrically monitoring whether the average global vector is in their intersection.

Unfortunately, this method suffers from similar issues as the *Covering Spheres* method, and cannot be applied on some basic functions, thus isn't feasable [**?**]

D. The Convex Bound Method – The Vector Scheme

The *Covering Spheres* method and the *Convex Decomposition* method turned out to be impractical albeit their mathematical purity. In need of more computationally lightweight and consistent monitoring approach, the *Convex Bound* method was proposed [**?**].

The *Convex Bound* method simply bounds the monitored function by a convex function, so the convex bound serves as the convex safe zone: when monitoring $f(v) \leq T$, an upper bound convex c has to be found so for all v, $f(v) \leq c(v)$. And the new monitoring objective becomes:

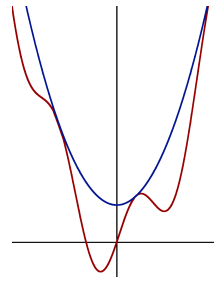

Fig. 2: $x^2 + 10$ is an upper convex bound of $x^2 + 10sin(x)$

$$c(v) \leq T \tag{5}$$

The same goes for lower bound threshold monitoring – It's done by bounding f from below by a concave function. Accordingly, the distributed function monitoring is done on the convex bound functions, which is far simpler due to the convexity property.

The process of bounding a function with another convex function has to be done carefully. First, we'd like the convex bound to be $tight$, so less data is lost in the bounding process. Secondly, the state of the initial global vector has to be considered; given a function f to monitor and a starting global vector v_0, it's probably better to engineer a convex bound function c which satisfies $f(v_0) = c(v_0)$.

After bounding the function from above and below by a convex function, the monitoring is done so whenever a local vector gets out of the convex bound, a violation is raised. The monitoring scheme tries to resolve this violation or raises a *Full Sync*, which updates the upper and lower convex bound functions. In [**?**] the *Vector Scheme* was proposed, which resolves the violations by "balancing" the local vectors (requires high dimensional vectors to be transmitted).

Here, we propose the *Value Scheme* and the *Distance Scheme* which resolve the violations by sending one single scalar. Another monitoring scheme we present is the *Sketched Data Scheme* which fuses the previously developed *Vector Scheme* and the one-dimensional schemes we developed.

III. Vector Scheme

The *Vector Scheme*'s main idea is to balance the server's data vectors whenever a local vector gets out of the function's convex bound. The *Vector Scheme* tries to balance the *violated server* with other server's data vectors. It is done by incorporating *slack vectors*, namely, $server_i$ maintains a slack $\overrightarrow{s_i}$. At all times the sum of the slacks is zero: $\sum \overrightarrow{s_i} = 0$.

For an upper-bound threshold, the local constraints at the servers is that $c(v_i + s_i) > T$. It follows that a server raises a violation and initiates a communication channel with *the coordinator* if $c(v_i + s_i) > T$, i.e. the local vector plus the slack exceeds the threshold. This ensures that whenever all the local constraint are being held, the global threshold inequality (**??**) is held as well.

Proof due to convexity of c, sum of slacks is zero and $c(v_i + s_i) \leq T$ for all i:

$$\begin{aligned} c(v) &= c\left(\frac{1}{n}\sum_{i=0}^{n} v_i\right) = \frac{1}{n}c\left(\sum_{i=0}^{n}(v_i + s_i)\right) \\ &\leq \frac{1}{n}\sum_{i=0}^{n} c(v_i + s_i) \leq \frac{1}{n}(n \cdot T) = T \end{aligned} \tag{6}$$

When a violation occurs, and a certain server upholds $c(v_i + s_i) > T$, proof (**??**) isn't necessarily right so a *violation resolution protocol* has to be initiated.

A. Violation Resolution

In the *violation resolution* phase, the slack vectors are balanced so $c(v_i + s_i)$ at the violated server would get inside the convex zone $\{v \mid c(v) \leq T\}$. When a server detects a local violation, it sends its local vector $(v_i + s_i)$ to *the coordinator*, which then gradually polls other servers for their local vectors. Let $(k-1)$ be the number of polled servers; denote $\widetilde{v} = \frac{1}{k}\sum(v_i + s_i)$ as the average of vectors of the polled servers and the violated server; then, whenever this average is inside the convex zone, i.e. $c(\widetilde{v}) \leq T$, the violation can be resolved, and no more servers have to be polled: *the coordinator* sends this average vector – $\widetilde{v}$ to the polled servers as well as the violated server, which update their slack vector to be $s_j \leftarrow -v_j + \widetilde{v}$.
After this update, the sum of the slack vectors is still zero, and $c(v_i + s_i) \geq T$ at all the servers, so (**??**) is correct again, and the global threshold inequality (**??**) holds again.
If all the servers are polled and still $\widetilde{v}$ isn't inside the convex zone $c(\widetilde{v}) \geq T$, a *Full Sync* has to be done; since *the coordinator* knows each server's $(v_i + s_i)$, *the coordinator* can calculate the global vector v, and hence $f(v)$. Then, the upper bound's threshold and lower bound's threshold are reset to $(1 \pm \varepsilon)f(v)$ and the monitoring continues with *the coordinator* notifying the servers of the new bounds and their local vector – v and their new slack – the zero vector.

B. Communication Bandwidth

Considering the data vectors are of very high dimension, this scheme sends whole high dimensional vectors whenever even the slightest violation occurs. Therefore, even though the *Vector Scheme* is better than previous monitoring scheme, its still quite wasteful in communication bandwidth.

IV. Value Scheme

The *Value Scheme* is based on the *Convex Bound* method, so it also bounds the monitored function f by a convex function c so it gets between the threshold and the function's value.
The *Value Scheme* is a distributed monitoring scheme which reduces the bandwidth from sending a whole high dimensional vector to sending just one scalar. The transmitted scalar represents the *value* of the convex function c.
The *Value Scheme* maintains local scalar slack values; $server_i$ maintains the scalar λ_i. Also, at all times we'd enforce that the sum of the slacks is zero: $\sum \lambda_i = 0$. Here, the local server's constraint is whether $c(v_i) + \lambda_i \leq T$; hence, if all the local constraints are being held, the global threshold inequality (**??**) is held as well:

$$\begin{aligned} c(v) &= c\left(\frac{1}{n}\sum_{i=0}^{n} v_i\right) \leq \frac{1}{n}\sum_{i=0}^{n} c(v_i) \\ &= \frac{1}{n}\sum_{i=0}^{n}(c(v_i) + \lambda_i) \leq \frac{1}{n}(n \cdot T) = T \end{aligned} \tag{7}$$

When a violation occurs, i.e. $c(v_i) + \lambda_i > T$, proof (**??**) cannot longer hold, thus a violation resolution protocol has to be initiated.

A. Violation Resolution

The violation resolution protocol goes as follows: the violated server $server_i$ sends its local value $(c(v_i) + \lambda_i)$ which exceeds the threshold T. *The coordinator* tries to "balance" this scalar value by gradually polling other servers for their $(c(v_i) + \lambda_i)$ value. Let $(k-1)$ be the number of polled servers, then, denote $\widetilde{\lambda} = \frac{1}{k}\sum(c(v_i) + \lambda_i)$ the average of the local values plus the slacks of the polled servers and the violated server; so when $\widetilde{\lambda} \leq T$, the violation is resolved by sending the value $\widetilde{\lambda}$ to the polled servers and the violated server, which in turn, set their local scalar slack to $\lambda_j \leftarrow -c(v_j) + \widetilde{\lambda}$. Its important to note that the sum of the slacks is still zero after the resolution, and $c(v_i) + \lambda_i \leq T$ at all the servers as well, so proof (**??**) can hold again.
In case all the servers are being polled without being able to "balance" the slack, a *Full Sync* has to be done, which is very expensive regrading the communication bandwidth.

B. Communication Bandwidth

In the *Value Scheme*, the whole distributed monitoring is reduced to tracking one scalar value. Instead of transmitting a whole high dimensional vector as in the *Vector Scheme*, the transmitted data is just one scalar. Thus, we'd expect having much less communication bandwidth in this monitoring scheme.

V. Distance Scheme

The *Distance Scheme* is one more *distributed monitoring scheme* that relies on passing a single scalar when communicating. This scheme is also based on the *Convex Bound* method: the monitored function f is bound by a convex function c so the monitoring objective is as (**??**), to monitor whether $c(v) \leq T$. Of course, a lower-bound monitoring is done simultaneously with a different concave bound function. The transmitted scalar represents the *distance* to the convex function's boundary from the local vectors at the servers.
This scheme is based on the *Distance Lemma* which is proved below.

A. The Distance Lemma

The *Distance Lemma* states that the average of points is inside a convex body, if the sum of the distances to the surface of the points inside the convex body is greater than

the sum of the distances to the boundary of the points outside of the convex body. This *Distance Lemma* is the basis of the *Distance Scheme*.

The *Distance Lemma*: Let $\{v_1...v_n\}$ be vectors and let C be a convex body. If the sum of the distances to the boundary of C of the vectors inside C is greater than the sum of the distances to the boundary of the vectors from outside, then the average vector $\frac{1}{n}\sum v_i$ is inside the convex body.
proof:

1) Let $p_1, ..., p_k$ be the points inside the convex body C and $l_1, ..., l_k$ their distances to the boundary.
2) Let $q_1, ..., q_m$ be the points outside the convex body C and $d_1, ..., d_m$ the distance vectors from the boundary to the points. Let $c_1, ..., c_m$ be the points on the boundary which are the sources of the distance vectors $d_1, ..., d_m$ respectively.
3) Let $d = \sum d_i$ and $l = \sum l_i$
4) Assume $l \geq \sum ||d_i||$, so $l \geq ||d||$
5) It's needed to prove that the average point is inside the convex body, i.e., prove that $\frac{1}{k+m}(\sum p_i + \sum q_i) \in C$:

$$\begin{aligned}&(p_1 + ... + p_k) + (q_1 + ... + q_m) = \\ &(p_1 + ... + p_k) + (c_1 + d_1) + ... + (c_m + d_m) = \\ &(p_1 + ... + p_k) + d + (c_1 + ... + c_m) = \\ &\left(p_1 + \frac{l_1 d}{l}\right) + ... + \left(p_k + \frac{l_k d}{l}\right) + (c_1 + ... + c_m)\end{aligned} \tag{8}$$

$c_i \in C$ by definition – c_i is on the boundary of C. Also, $p_i + \frac{l_i d}{l} \in C$ because $l \geq ||d||$ and l_i is the distance of p_i to the boundary.
Since an average of points inside a convex body is also inside the convex body, $\frac{1}{k+m}(\sum p_i + \sum q_i) \in C$ ■

B. Distributed Monitoring

The *Distance Scheme*'s monitoring is based on the distance value of the local vectors to the function's convex bound geometric boundary. The *Distance Scheme* literally monitors distributively whether the sum of the distances from inside to the boundary is greater than the sum of the distances to the boundary from the outside. For convenience, denote the distance to the boundary from inside the convex body as negative, and the distance from outside as positive, therefore, the *Distance Scheme* monitors whether the sum of the distances is negative.
Like the *Value Scheme*, the *Distance Scheme* maintains slack scalars that would help "balance" the distances from the boundary between servers as the data changes. Let $server_i$ maintain the scalar λ_i. As in *Value Scheme*, at all times the sum of the slacks is zero: $\sum \lambda_i = 0$.
Denote d_i the distance of $server_i$'s vector to the boundary, then the local server's monitoring constraint is whether $d_i + \lambda_i \leq 0$. Therefore, if all the local constraints hold, the global vector's distance is negative and by the *Distance Lemma*, the global threshold inequality (**??**) holds:

$$\sum d_i = \sum (d_i + \lambda_i) \leq n \cdot 0 = 0 \tag{9}$$

The sum of the distances is negative, thus the average vector is inside the convex safe zone $\{v \mid c(v) \leq T\}$.
Whenever $d_i + \lambda_i > 0$ at a certain server, a violation occurs and a *violation resolution protocol* begins.

C. Violation Resolution

When the local vector at a server changes so $d_i + \lambda_i > 0$, a violation occurs. In order to resolve the violation, the λ_i slacks has to be "balanced" so $d_i + \lambda_i \leq 0$ at the violated server as well as other servers where some slack would be "borrowed". Firstly, in order to resolve the violation, the violated server sends its $(d_i + \lambda_i)$ value to the *Coordinator*; afterwards, *the coordinator* gradually polls other servers for their $(d_i + \lambda_i)$ value. Let $(k-1)$ be the number of polled servers, denote $\widetilde{\lambda} = \frac{1}{k}\sum(d_i + \lambda_i)$ the average of distances plus the slacks of the polled servers and the violated server; then when $\widetilde{\lambda} \leq 0$, the violation can be resolved: *the coordinator* sends the single value $\widetilde{\lambda}$ to the polled servers and the violated server, where they update their slacks so $\lambda_j \leftarrow -d_j + \widetilde{\lambda}$. Note that the sum of slacks is still zero after the update, and at each server the local constraint holds, $d_i + \lambda_i \leq 0$, so proof (**??**) holds again.

D. Communication Bandwidth

The monitoring is done solely by the value of the distance. A scalar value is transmitted between the *Coordinator* and the servers, thus we'd expect much less communication bandwidth than the *Vector Scheme*.

VI. Value Scheme vs. Distance Scheme

Both *Value Scheme* and *Distance Scheme* are distributed monitoring schemes which are reduced to tracking just one scalar value. In the *Distance Scheme* its the sum of the distances to the convex bound's geometric boundary, and in the *Value Scheme* its the average of the convex function's value.
A good question that arises is which monitoring scheme is better? which one uses less communication bandwidth? Is there a connection between the schemes?
In [*] its proposed there's a linear correlation between the

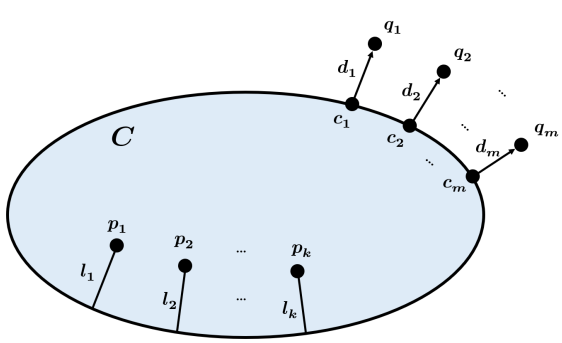

Fig. 3: The *Distance Lemma*

The average of $p1...p_k, q_1...q_m$ is inside the convex body C if $\sum l_i \geq \sum ||d_i||$.

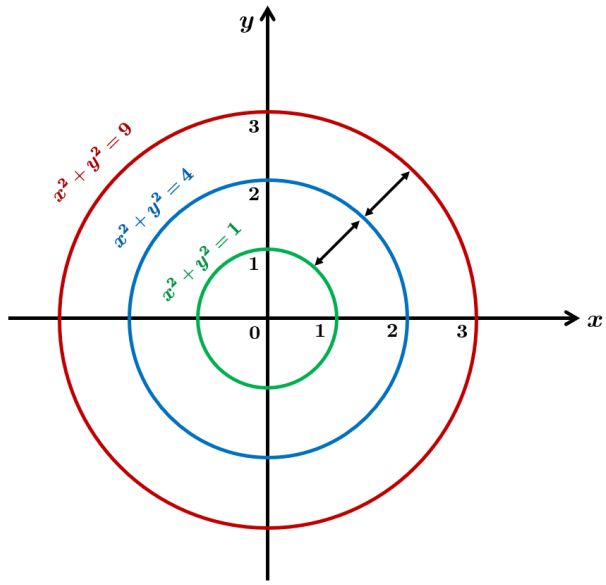

Fig. 4: Norm $L2$ Squared distance and value behaviour

The value of the function $||v||^2$ quadratically grows in relation to the distance.

distance to the geometric representation of a function and the value of the function when the distance is small. Hence, we'd expect similar results of the schemes.

Nonetheless, since we deal with convex functions, the value grows faster than the function's distance. A good example is the function $f(x) = x^{100}$ which when monitoring $f(x) \leq 1$, the function's value changes rapidly outside the boundary as x changes a bit; so the distance changes much slower than the value. Accordingly, we'd expect the *Distance Scheme* to be better on those convex function whose value grows rapidly in relation to the distance (vice versa cannot occur since the function is convex). This behaviour is shown in the experiment done in subsection **??** on the function $f(v) = ||v||^2$, where *Distance Scheme*'s communication bandwidth is much less than the *Value Scheme*. The quadric relation of the value-distance is demonstrated in Fig. **??**.

Even though in convex functions the growth of the distance progresses slower than its value, it doesn't necessarily imply that the distance scheme is better. The function's value growth is two sided – when a function grows fast on one direction, it may decay fast on the reverse direction. For example, when monitoring the convex condition $f(x) = y - x^2 \geq 2$, the results aren't conclusive. As shown in Fig. **??** and Fig. **??**, there are cases of false negative violations which occur only in one distributed monitoring scheme and not the other; the global average vector remains inside the convex bound $f(x) \geq 2$ while a false alarm can be raised.

Another important factor is that the *Value Scheme* is computationally mathematically whole, while the *Distance Scheme* may require some approximations; the problem of finding the distance to a convex surface from outside is mathematically closed using convex optimization techniques [**?**]. On the other hand, finding the distance from inside to the convex surface isn't always solvable [**], so unfortunately the *Distance Scheme* isn't applicable on all possible functions that require distributed monitoring.

VII. Sketched Data Scheme

The *Sketched Data Scheme* is a gradual integration of the *Vector Scheme* and either the *Value Scheme* or the *Distance Scheme*.

This scheme behaves like the *Value Scheme* or the *Distance Scheme* up until a violation which cannot be resolved occurs and a *Full Sync* has to be done. Instead of a *Full Sync*, *the coordinator* polls the servers for a *sketch*, i.e., a lossy compression of their change vector of the data; the average of the sketches is combined and sent to the servers, which infer some information from it which might help continuing the monitoring without invoking an expensive *Full Sync*. For the sketch function, multiple functions can be used, such as sending the most dominant elements of the change-vector, the most dominant DCT parameters of the change-vector and the most dominant PCA parameters etc.

The better the sketch represents the change vector, the more beneficial information the servers will deduce which will help overcome the violation that occurred in the one-dimensional scheme (*Value Scheme* or *Distance Scheme*).

Practically, the *Sketched Data Scheme* performs a softer more gradual *Full Sync*; if this partial *Full Sync* still won't allow the one-dimensional distributed monitoring scheme to be continued, the dimension of the sketch will be raised so the lossy compressed sketches would be more representative of the global vector, hopefully allowing the one dimensional scheme to continue.

A. Distributed Monitoring

The algorithm of the *Sketched Data Scheme* acts as follows:

1) Choose a one-dimensional distributed monitoring scheme (*Value Scheme* or *Distance Scheme*), note it σ.
2) Distributively monitor the global monitored function according to σ.
3) When a *Full Sync* has to be invoked because of σ, each server sends a sketch of dimension d of the change vector to *the coordinator*. Initially $d = const$.
4) *The coordinator* sends the average of the sketches to the servers in a compressed form according to the sketch function.
5) The servers add the average of sketches to their reference vector, and set their change vector to be the "error" of their compression over their original change vector.
6) Try continuing monitoring by σ, if it doesn't immediately raise a *Full Sync* – set $d = const$ and go to (**??**) to continue monitoring.
   Otherwise, if a *Full Sync* was raised immediately, multiply d by two, and go to (**??**) to try resolving the violation.

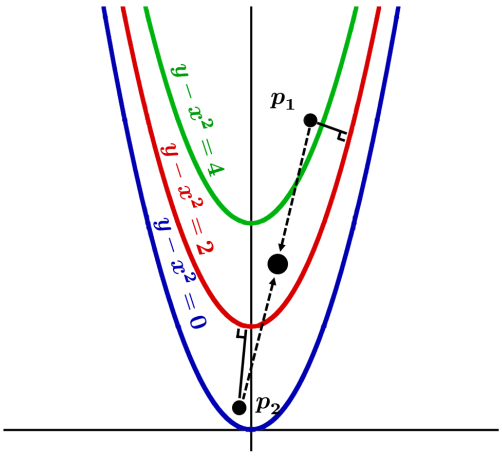

Fig. 5: *Value Scheme* beats *Distance Scheme*

When monitoring $y - x^2 \geq 2$, the distance to the boundary $f(x, y) = y - x^2 = 2$ from the outside point p_2 is greater than the distance from the inside point – p_1, though $\frac{1}{2}(f(p_1) + f(p_2)) \geq 2$, so the *Value Scheme* won't raise a violation, while the *Distance Scheme* will raise a violation which is a *false alarm*.

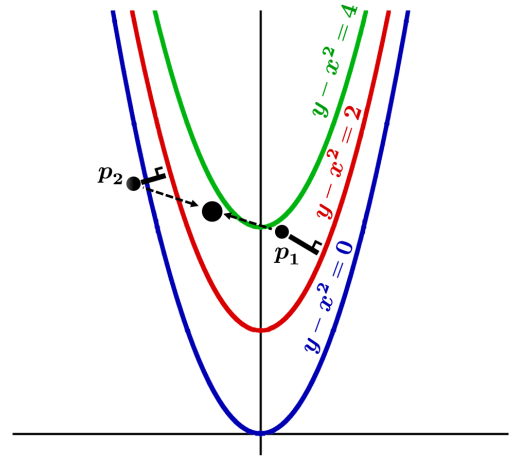

Fig. 6: *Distance Scheme* beats *Value Scheme*

When monitoring $y - x^2 \geq 2$, the distance to the boundary $f(x, y) = y - x^2 = 2$ of the inside point p_1 is greater than the distance of the outside point p_2, however, $\frac{1}{2}(f(p_1) + f(p_2)) < 2$, so the *Value Scheme* will raise a false-negative violation, while the *Distance Scheme* correctly won't raise a violation.

7) If d reaches the value of quarter of the data-vector dimension, invoke a global *Full Sync*, set $d = const$ and go to (**??**) to continue monitoring.

B. Violation Resolution

Here we'll elaborate about the violation resolution phase of the *Sketched Data Scheme*. The violation resolution of the *Sketched Data Scheme* is invoked when the one-dimensional monitoring scheme fails.

1) Let n be the number of servers. Let r_0 be their reference vector and $ch_1 ... ch_n$ be their data's change-vectors of the servers.
2) Let d be the chosen dimension of the violation resolution.
3) Let sk_d be a lossy sketch function of dimension d. Let $sk_d(ch_i)$ be the sketch of the change vector of the i-th server, and $\varepsilon_d(ch_i)$ be the "error" of the sketch function so $ch_i = sk_d(ch_i) + \varepsilon_i(ch_i)$ for all $server_i$.
4) Each server sends its $sk_d(ch_i)$ to *the coordinator* in a compressed form.
5) *The coordinator* calculates the average of the sent vectors $\widetilde{sk_d} = \frac{1}{n} \sum sk_d(ch_i)$ and transmits it to the servers.
6) Each server $server_j$ updates its reference vector to be $r_0 \leftarrow r_0 + \widetilde{sk_d}$. The change vectors are updated as well: $ch_j \leftarrow \varepsilon_j(ch_j)$.
7) Try continuing the one-dimensional distributed scheme.
8) If the one-dimensional scheme immediately tries to invoke a *Full Sync* again, raise the sketch dimension to $2 \cdot d$ up to $\frac{n}{4}$.
9) If a violation reoccurs when $d > \frac{n}{4}$ invoke a *Full Sync*.

It's important to note that at all times before and after the violation, the global vector remains the same: $v = \frac{1}{n} \sum v_i = \frac{1}{n} \sum (r_0 + ch_i)$. Moreover, the "error" of the sketch function is taken into account so there's no probabilistic nature to this scheme. The more the sketch function represents the change vector, the better the local vectors of the servers estimate the global vector.

C. Communication Bandwidth

The *Sketched Data Scheme* actually makes the *Full Sync* more gradual. Instead of sending the whole data vectors, it sends it bit by bit, so when the partially average vector is enough for the *Value Scheme* or the *Distance Scheme* to continue monitoring on its own – there's no need for more communication to be made.

In addition, the sketches are done on the change vector, so the servers embed the data they get from round to round into the reference vector, hence the "information" the server receives gets accumulated; therefore, when a *Full Sync* has to be invoked, already half of the global change vector is accumulated by the servers, so what's left is only the other half, which make the *Full Sync* more efficient, requiring half as much communication bandwidth.

Also, since the dimension of the sent vector doubles itself from round to round, the latency of the *Sketched Data Scheme* is logarithmic to the dimension of the data.

VIII. Experimental Results

In order to assess the productivity of the distributed monitoring schemes proposed, we tested the communication bandwidth of the data sent between the servers and *the coordinator* both on real-world diverse data and synthetic scheme-oriented data.

As for the distributed dynamic monitored data, we used a bag-of-words vector of high dimension constructed of occurrences of tokens at a moving sliding window.

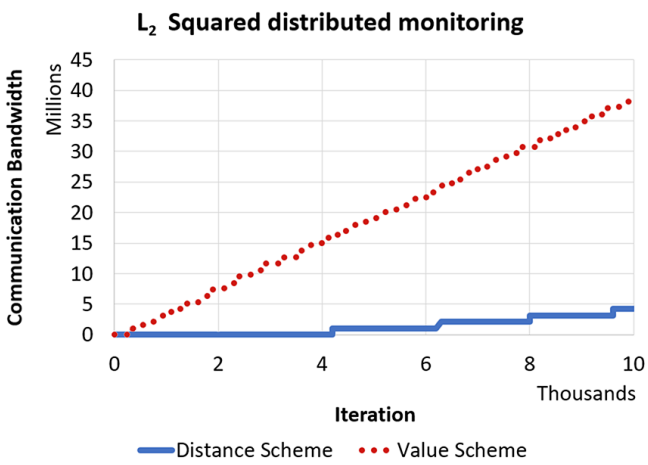

Fig. 7: Distributed Monitoring of the L_2 Squared Function

Distance Scheme uses much less communication bandwidth than *Value Scheme* when monitoring norm L_2 squared function.

The servers data is diverse: one is created out of blogs from *bloggers.com* while other is fed of tweets posted online, other from Amazon reviews, one more from Reddit comments etc.

A. Oracle Scheme and Naive Scheme

One of the needs of comparing between the monitoring schemes and assessing whether the monitoring schemes proposed really are economical bandwidth-wise, is offering a lower bound and an upper bound for distributed monitoring schemes.

The obvious communication-bandwidth upper bound is the *Naive Scheme* which simply sends each data change at a server to *the coordinator*, so virtually, the global vector is known, and the computation isn't distributed but centralized. This method obviously isn't very bandwidth efficient, though its a good upper bound for comparison.

For the lower bound of the communication bandwidth, we propose the *Oracle Scheme* which is like an oracle who does not have false negative violations. Simply, no communication is done, apart from whenever a *true violation* occurs and a *Full Sync* is invoked. Let μ be the last "known" value of $f(v)$, then, a *true violation* is when $f(v) > (1+\varepsilon)\mu$ or $f(v) < (1-\varepsilon)\mu$ (for simplicity, assume $\mu \geq 0$). Naturally, all the changed data is sent at a *Full Sync*, and the value of μ is updated as well as the thresholds values.

In other words, *Oracle Scheme* transmits data only when it's truly needed to be passed – when a *Full Sync* must be done. Of course this scheme isn't realistic, since a server knows nothing about its neighbours, so the global vector isn't known. False violation obviously have to occur from time to time.

B. Norm L_2 Squared

The L_2 norm is widely used in multiple scenarios, either for (...) [*] or for (...) [**].

Here, we monitored the L_2 norm squared, namely $f(v) = ||v||^2$. One of the reasons is to confirm that this function requires less communication bandwidth using the *Distance Scheme* in contrast to the *Value Scheme*.

1) Lower and Upper Bounds: Finding the convex bound of the function isn't difficult since it's convex; the upper bound is the function itself, and the lower bound's concave function is the tangent plane to the function at the closest point from the global vector. Computing the distance from a point to the boundary also is fairly easy – both to the sphere and to the tangent plane.

2) Experiments: In order to check the difference between the *Value Scheme* and the *Distance Scheme* we constructed one hundred servers with dynamic vectors in dimension 10,000. The changes were of small steps in all the 10,000 dimensions. The bandwidth results over time are presented in Fig. **??**. The *Distance Scheme* resulted in about eight times less transmitted data.

Since we started with zero vectors, we monitored the function by bounding it additively (± 10) instead of a multiplicative $(1 \pm \varepsilon)$.

C. Entropy

The Entropy function is widely used in computer science applications. Whether its tracking a distributed denial of service attack and network anomalies [**?**], medicine applications [**?**] or detection of physical activity using wearable sensors [**?**].

The entropy function of the probability-vector v of dimension n is defined as follows:

$$f(v) = f(v_1, ..., v_n) = \sum_{i=0}^{n} -v_i \cdot ln(v_i) \tag{10}$$

1) Lower and Upper Bounds: Applying the convex-bound method on the entropy function requires finding a convex bound from above and a concave bound from below. As for the upper bound, since the function is concave, a good upper bound is the tangent plane at the closest point. For the lower bound, seemingly, because the function is concave, it should

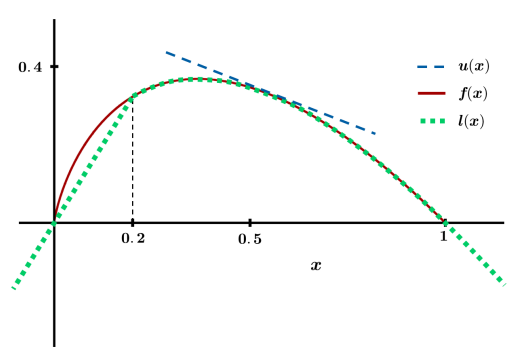

Fig. 8: Entropy Function's Upper and Lower Bounds

$f(x) = -xln(x)$ is the entropy function. A convex upper bound is the tangent plane at a point as in $u(x)$. A concave lower bound is shown in $l(x)$. The extension point $p_0 = 0.2$ is chosen for illustration.

be its own concave bound, which isn't fully correct. Notice we work not on the data vectors but on the reference point plus the change vector, so parameters can become negative, which isn't defined on the entropy function. Consequently, we'd have to "extend" the entropy function to the negative part. As suggested in [**?**], we bound the entropy function by itself, but below the point where the probability is $p_0 = \frac{1}{const}$ we take the line that starts at p_0 and passes at the origin of the coordinate system. A sketch of an instance of an upper bound and lower bound of the entropy function is shown in Fig. **??**.
After finding the convex upper bound and the lower concave bound, distributed monitoring of the entropy can be done by applying the *Value Scheme* or the *Vector Scheme* , though the *Distance Scheme* requires being able to calculate distances to the bounding function's surface.

2) Distance to Bounds: The main challenge is computing the distance from inside the concave lower bound.
The convex upper bound is a tangent plane. Finding the distance from a point to the tangent plane's surface has a closed mathematical solution.
As for the concave lower bound, finding the distance to the entropy's bound surface from outside is solvable by convex optimization [**?**], but from inside, finding the L_2 norm isn't easy. So instead of computing the L_2 distances, we compute L_1 distances so the *Distance Scheme* is done on L_1 distances. Note that the *Distance Lemma* is correct on any norm of distances since the norm it uses requires only the triangle inequality.
Since the entropy's lower bound function is concave and permutation invariant, finding its L_1 distance can be calculated by a basic binary search on the value of L_1 done by climbing the derivation function.
Unfortunately, this method for finding the L_1 distance to the surface is quite costly in time for high dimensional vectors, thus allowing us to experiment only on low dimensional data using the *Distance Scheme*.

3) Experiments: ....

D. Inner Product

In order to adjust the inner-product function to the distributed monitoring model, we treat the inner-product as a function whose input is a vector of even dimensionality. Denote $[x \circ y]$ as the concatenation of vectors x and y. Assume x and y are of the same dimensionality, then the inner product function is defined as follows:

$$f([x \circ y]) =< x, y > \tag{11}$$

The inner-product function is extensively used in mathematics and computer science either as a similarity measure of data [*] or as a basic component of other functions: it's used in data mining applications [*] and machine learning softwares [*] and also in ... [*].

1) Lower and Upper Bounds: A good trick presented in [**?**] is representing a function as a difference of two convex functions, $f(v) = c_1(v) - c_2(v)$; then, binding it by a convex (concave) function requires solely bounding the negative (positive) part from above (below) by a tangent plane.
As for the inner product, it upholds that $< x, y > = 0.25(||x + y||^2 - ||x - y||^2)$ which is a difference of two convex functions.

2) Distance to Bounds: After having a closed mathematical representation to the upper and lower bound, the distance from a point to the function's surface can be found using the method of Lagrange multipliers which yield a closed mathematical solution.

3) Experiments: ....

IX. Conclusions

....