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.

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, allowing for some monitoring trade-offs to be made, depending on the sketch size.

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 large distributed dynamic graphs [1] or a distributed sensor network [2], 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 [3]; a good example is internet of things objects which are operated on batteries, hence sending data via a communication channel is very costly, thus should be minimized [*]. Furthermore, in the *Big Data* era, when data is of very high dimensionality and changes rapidly, sending the whole 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 [4].

Previous works were made on linear functions [5], 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 [6][**][***], monitoring the inner-product value of distributed dynamic vectors [7], and monitoring the number of triangles of dynamic graphs held in distributed servers [8].

It should be noted that the classic approach to monitor distributed dynamic data is to send the data changes at a certain frequency to a central server, called *the coordinator*. This method may miss peaks of critical global data-changes, and also may be more expensive communication-wise – not only data will be sent even when little changes are made, but the whole big dimensional data changes will be transmitted which might be extremely expensive [*].

The distributed function approximation problem is defined as follows [7]:

1) There are n servers and a coordinator.
   $server_i$ maintains its dynamic *local vector* v_i.
   The *global vector* is the average of the local vectors:

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

2) A function f is monitored over the *global vector* v so it is ε-approximated with 100% confidence.
   Let the estimation be the value μ and $f(v) \geq 0$, than at

all times:

$$(1-\varepsilon)\mu \leq f(v) \leq (1+\varepsilon)\mu$$

The function's approximation problem can is reduced to *threshold monitoring problem* [7] where one monitors whether the function's value crosses a certain threshold. specifically, let T be the threshold's value, than the general upper-bound monitoring objective is to determine whether:

$$f(v) \leq T \tag{1}$$

In turn, this threshold monitoring can be reduced to a *geometric monitoring problem* [9], 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*. This geometric *safe zone* approach is the fundamental idea behind previous distributed monitoring techniques.

A. Contributions

1) Introducing multiple innovative distributed monitoring schemes which avoid sending big dimensional data unless its crucially critical.
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, than 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;e.g., we managed to prevent having the probabilistic nature of the sketches affect the distributed monitoring scheme having a false-negative 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 convex, the basic algorithm for distributively monitoring their value is fairly easy. A work about linear functions such as the discributed count problem was done at [5]. Though, things get more sophisticated when dealing with non-linear functions, where novel approaches had to be developed.

***** Should I add something here???? *****

B. The Covering Spheres Method

C. The Convex Decomposition Approach

D. The Convex Bound Method – The Vector Scheme

(4) Convex Bound Method – Vector Scheme The monitoring is done by bounding the function f with a 'convex bound' function [*] – a function c which bounds the function from above (below) and used for the upper (lower) threshold monitoring. Given a function f to monitor and a starting global vector v_0, it was proposed to approach the problem by finding a convex function c which bounds the function f i.e. for all v, $f(v) \leq c(v)$ and preferably $f(v_0) = c(v_0)$ [**].

$$f(v) \leq T \tag{2}$$

and since $f(v) \leq c(v)$ for all v, we'd monitor whether:

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

III. Vector Scheme

The *Vector Scheme*'s idea is to balance the server's data vectors whenever a local vector gets out of the function's convex bound. The *Vector Scheme* would try to balance the *violated server* with other server's data vectors. It is done by incorporating *slack vectors*, namely, $server_i$ would maintain a slack $\overrightarrow{s_i}$. It's important to note that the *Vector Scheme* makes sure that at all times the sum of the slacks is zero: $\sum \overrightarrow{s_i} = 0$. In order to take into consideration these *slacks*, a server raises a violation and initiates a communication channel with the coordinator if $c(v_i + s_i)$ exceeds the threshold; specifically, for an upper bound threshold, when $c(v_i + s_i) > T$. This ensures that whenever all the local constraint hold, the global constraint (3) holds. proof due to convexity of c, sum of slacks is zero and $c(v_i + s_i) \leq T$:

$$\begin{aligned} c(v) &= c\left(\frac{1}{n}\sum_{i=0}^{n} v_i\right) = \frac{1}{n} c\left(\sum_{i=0}^{n}(v_i + s_i)\right) \\ &\leq \frac{1}{n}\sum_{i=0}^{n} c(v_i + s_i) \leq \frac{1}{n}(n \cdot T) = T \end{aligned} \tag{4}$$

When a violation occurs, i.e. $c(v_i + s_i) > T$ at a certain server, (4) cannot longer be proven so a *violation resolution protocol* has to occur.

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. When a server detects a local violation, it sends its local vector $(v_i + s_i)$ to the coordinator, which polls other servers for their local vectors as well. When the average of those vectors is inside the convex zone, i.e. $c(E(v_i + s_i)) \leq T$ the resolution is almost done, and no more servers have to be polled. Let $(k-1)$ be the number of polled servers, then the coordinator sends the average vector — $\frac{1}{k}\sum(v_i + s_i)$ to the polled servers as well as the violated server, which update their slack to be $s_j \leftarrow -v_j + \frac{1}{k}\sum_i (v_i + s_i)$. Note that the sum of the slack vectors is still zero.

When all the servers are polled and the average vector still isn't inside the convex zone, 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 and lower bound 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 the whole vectors whenever even the slightest violation occurs. Therefore, even though the *Vector Scheme* is better than the naive monitoring scheme, its still wasteful in communication bandwidth.

IV. Value Scheme

The *Value Scheme* is a distributed monitoring scheme which reduces the bandwidth from sending a whole high dimensional vector to sending just one scalar. Though, due to the dimensional reduction, we'd expect more false alarms to occur than at the *Vector Scheme*, and thus require more *full syncs*. The scalar that will be passed will represent the *value* of the convex bound function.
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 constraint (3) will be held as well:

$$\begin{aligned} c(v) &= c\left(\frac{1}{n}\sum_{i=0}^{n} v_i\right) \leq \frac{1}{n}\sum_{i=0}^{n} c(v_i) \\ &= \frac{1}{n}\sum_{i=0}^{n}(c(v_i)+\lambda_i) \leq \frac{1}{n}(n\cdot T) = T \end{aligned} \tag{5}$$

When a violation occurs, i.e. $c(v_i) + \lambda_i > T$, proof (5) 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. 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 plus the violated server, then, when $\frac{1}{k}\sum(c(v_i)+\lambda_i) \leq T$ the violation will be resolved by sending $\frac{1}{k}\sum(c(v_i)+\lambda_i)$ to the polled servers and the violated server, which in turn, will set their local scalar slack to $\lambda_j \leftarrow -c(v_j) + \frac{1}{k}\sum(c(v_i)+\lambda_i)$. Its important to note that the sum of the slacks is zero after the resolution, and $c(v_i) + \lambda_i \leq T$ as well.
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. The transmitted data is just one scalar instead of a whole high dimensional vector, as in *Vector Scheme*. 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, treats the distributed monitoring problem as a *geometric monitoring problem*, where there's a geometric *safe zone*, $\{v \mid c(v) \leq T\}$ where it's valid for local vectors to be at. The scalar that will be passed in the monitoring represents the *distance* of the local vector at the server to the boundary of this convex bound.
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 distances to the surface of the points inside the convex body is greater than the sum of distances to the boundary of the points which are 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 set.
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 source of the distance vectors $d_1, ..., d_m$ respectively.
3) Let $d = \sum d_i$ and $l = \sum l_i$
4) Assume $l \geq d$
5) It's needed to prove the average vector is inside the convex body, i.e., prove that $\frac{1}{k+m}(\sum p_i + \sum q_i) \in C$:

$$\begin{aligned} &(p_1 + ... + p_k) + (q_1 + ... + q_m) = \\ &(p_1 + ... + p_k) + (c_1 + d_1) + ... + (c_m + d_m) = \\ &(p_1 + ... + p_k) + d + (c_1 + ... + c_m) = \\ &\left(p_1 + \frac{l_1 d}{l}\right) + ... + \left(p_k + \frac{l_k d}{l}\right) + (c_1 + ... + c_m) \end{aligned} \tag{6}$$

$c_i \in C$ 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* is based on the distance value of the local vectors to the convex 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 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 constraint (3) holds:

$$\sum d_i = \sum (d_i + \lambda_i) \leq n \cdot 0 = 0 \tag{7}$$

The sum of the distances is negative, thus the average vector is inside the convex set.
Whenever $d_i + \lambda_i > 0$ at a certain server, a violation occurs and a *violation resolution protocol* has to occur.

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 others servers for their $d_i + \lambda_i$ value. Let $(k-1)$ be the number of polled servers, than when $\frac{1}{k}\sum(d_i + \lambda_i) \leq 0$ the violation can be resolved, and the coordinator sends the single value $\frac{1}{k}\sum(d_i + \lambda_i)$ to the polled servers and the violated server. Then, the slacks are updated so $\lambda_j \leftarrow -d_j + \frac{1}{k}\sum(d_i + \lambda_i)$. Note that the sum of slacks is still zero after the update, and at each server the local constraint holds, $d_i + \lambda_i \leq 0$.

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. Sketched Change Scheme

The *Sketched Change Scheme* is an 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* (e.g. the one-dimensional scheme) up until a violation which cannot resolved occurs and a *Full Sync* has to be made. Instead of the *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 element of the vector, the most dominant DCT parameters of the 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.
Practically, the *Sketched Change 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 Change Scheme* acts as follows:

1) Choose a one-dimensional distributed monitoring 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, send a sketch of dimension d of the change vector to the coordinator from each server. Initially $d = const$.
4) The Coordinator sends the average of the sketches to the servers.
5) The servers add the average of sketches to their reference vector, and set their change vector to be the loss of their compression over their original change vector.
6) Set the one dimensional scalar slack to zero
7) Try continuing the one dimensional scheme, if it immediately raises a *Full Sync* multiply d by two, otherwise continue monitoring – go to (2) and set $d = const$.
8) if d is half of the data-vector dimension, invoke a *Full Sync*, set $d = const$ and continue monitoring – go to (2).

B. Violation Resolution

The violation resolution of the *Sketched Change Scheme* is invoked when the one-dimensional monitoring scheme fails.

1) Let n be the number of servers. Let r be their reference vector and $ch_1 ... ch_n$ be their data's change-vectors of the servers respectively.
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.
5) The coordinator calculates the average of the sent vectors $\overline{sk_d} = \frac{1}{n}\sum sk_d(ch_i)$ and transmits it to the servers.
6) Each server $server_j$ will update its reference vector to be $r \leftarrow r + \overline{sk_d}$ (the reference vector is the same at

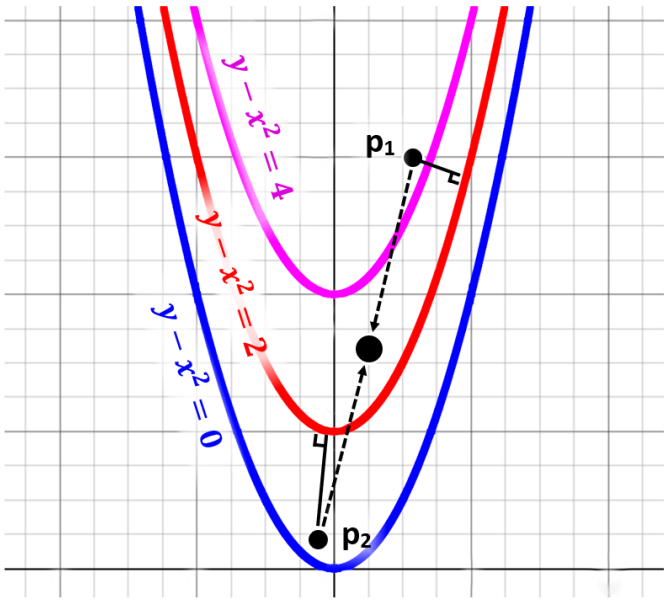

Fig. 1: *Value Scheme* beats *Distance Scheme*

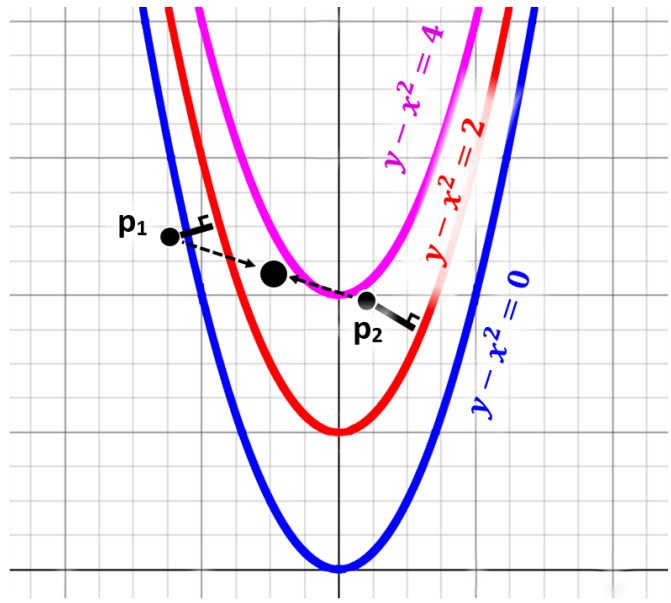

Fig. 2: *Distance Scheme* beats *Value Scheme*

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

all servers). The change vectors are updated as well: $ch_j \leftarrow \varepsilon_j(ch_j)$.

7) Set the value/distance scalar slacks of the servers to zero and 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 $\frac{n}{2}$.
9) if a violation reoccurs when $d \geq \frac{n}{2}$ invoke a *Full Sync*.

It's important to note that at all times before and after the violation, the global vector remains the same: $v = \frac{1}{n}\sum v_i = \frac{1}{n}\sum(r + ch_i)$. 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 better, the local vector of the server estimates better the global vector.

C. Communication Bandwidth

The *Sketched Change Scheme* actually makes the *Full Sync* more gradual. Instead of sending the whole data vectors. It sends it bit by ....

VII. 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 theres a connection between the distance to a function and the value of the function. moreover, when the distance is small, there's a linear correlation between the function's value and the distance.
Nonetheless, since we are dealing with convex functions, the value grows faster than the function's value. A good example is the function $f(v) = (\sum v_i)^{100}$ which when monitoring $f(v) \leq 1$, the function's value changes rapidly outside the boundary, whereas the distance changes regularly. This behavor is showed in the experiment done in section [**] on the function $f(v) = ||v||^2$.
Even though in convex functions the distance grows faster than its value, it doesn't necessarily imply that the distance scheme is better. for example, monitoring the convex condition $y - x^2 \geq 2$, the results aren't conclusive. As shown in (....) (add paintings) (.....) 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 the convex bound from outside is mathematically closed using convex optimization

techniques. On the other hand, finding the distance from inside to the convex body isn't always solvable [**], so the *Distance Scheme* isn't applicable in all the possible functions that require distributed monitoring.

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 in a moving sliding window in the servers. The servers data is diverse: one is created out of blogs from *bloggers.com* while other is fed of tweets posted online 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. This method obviously isn't very bandwidth efficient, though its a good upper bound. For the lower bound of the communication bandwidth, we propose the *Oracle Scheme* which is like an oracle which knows it all. Simply, whenever a true violation occurs, i.e. $f(v') > (1+\varepsilon)f(v)$ or $f(v') < (1-\varepsilon)f(v)$ (for simplicity, $f(v) \geq 0$), a *Full Sync*is invoked. Of course this scheme isn't realistic, since a server knows nothing about its neighbours, so the global vector isn't known. In other words, data is transmitted only when its truly needed to be passed – a *Full Sync* must be done.

B. Norm L_2 Squared

The norm L_2 is used in multiple scenarios, as in [*] and [**]. Monitoring this function distributively isn't Obvious even though the function is convex.

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

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 convex function is the tangent plane to the function at the closest point to the global vector with norm squarred as the lower bound's threshold.

To check the difference between the *Value Scheme* and *Distance Scheme* we constructed one hundred servers with a vector in dimension 10,000. The servers dynamically change. The changes were of small steps in all the 10,000 dimensions. The bandwidth results over time are presented in figure [***]. For this experiment we monitored the function by bounding it (± 10) instead of a multiplicative $(1 \pm \varepsilon)f(v_0)$.

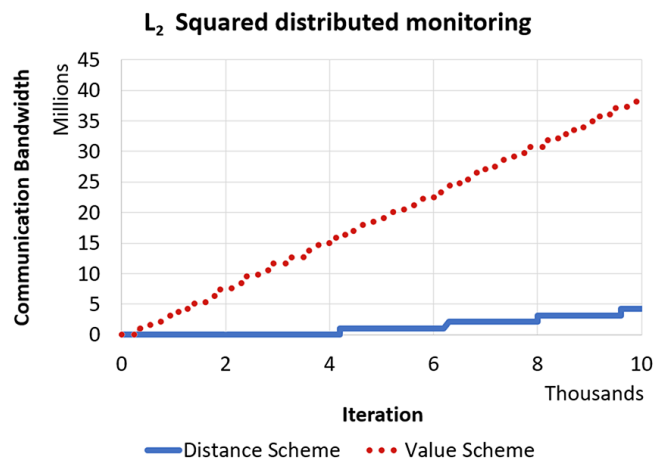

Fig. 3: *Distance Scheme* uses less communication bandwidth than *Value Scheme* when monitoring norm L_2 squared function

C. Inner Product

D. Entropy

IX. Conclusions

References

[1] A. McGregor, D. Tench, S. Vorotnikova, and H. T. Vu, "Densest subgraph in dynamic graph streams," in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 2015, pp. 472–482.

[2] S. Burdakis and A. Deligiannakis, "Detecting outliers in sensor networks using the geometric approach," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 1108–1119.

[3] N. Giatrakos, Y. Kotidis, A. Deligiannakis, V. Vassalos, and Y. Theodoridis, "In-network approximate computation of outliers with quality guarantees," *Information Systems*, vol. 38, no. 8, pp. 1285–1308, 2013.

[4] W.-L. Cheng, Y.-C. Kuo, P.-L. Lin, K.-H. Chang, Y.-S. Chen, T.-M. Lin, and R. Huang, "Revised air quality index derived from an entropy function," *Atmospheric Environment*, vol. 38, no. 3, pp. 383–391, 2004.

[5] R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication-efficient distributed monitoring of thresholded counts," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 2006, pp. 289–300.

[6] M. Gabel, D. Keren, and A. Schuster, "Anarchists, unite: Practical entropy approximation for distributed streams," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 837–846.

[7] M. Garofalakis, D. Keren, and V. Samoladas, "Sketch-based geometric monitoring of distributed stream queries," *Proceedings of the VLDB Endowment*, vol. 6, no. 10, pp. 937–948, 2013.

[8] G. Yehuda, D. Keren, and I. Akaria, "Monitoring properties of large, distributed, dynamic graphs," in *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 2017, pp. 2–11.

[9] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," *ACM Transactions on Database Systems (TODS)*, vol. 32, no. 4, p. 23, 2007.