A Generic Local Algorithm for Mining Data Streams in Large Distributed Systems

Ran Wolff, Kanishka Bhaduri, and Hillol Kargupta *Senior Member, IEEE*

***Abstract*— In a large network of computers or wireless sensors, each of the components (henceforth, peers) has some data about the global state of the system. Much of the system's functionality such as message routing, information retrieval and load sharing relies on modeling the global state. We refer to the outcome of the function (*e.g.*, the load experienced by each peer) as the *model* of the system. Since the state of the system is constantly changing, it is necessary to keep the models up-to-date.**

Computing global data mining models *e.g.* decision trees, k-means clustering in large distributed systems may be very costly due to the scale of the system and due to communication cost, which may be high. The cost further increases in a dynamic scenario when the data changes rapidly. In this paper we describe a two step approach for dealing with these costs. First, we describe a highly efficient *local* algorithm which can be used to monitor a wide class of data mining models. Then, we use this algorithm as a feedback loop for the monitoring of complex functions of the data such as its k-means clustering. The theoretical claims are corroborated with a thorough experimental analysis.

I. Introduction

In sensor networks, peer-to-peer systems, grid systems, and other large distributed systems there is often the need to model the data that is distributed over the entire system. In most cases, centralizing all or some of the data is a costly approach. When data is streaming and system changes are frequent, designers face a dilemma: should they update the model frequently and risk wasting resources on insignificant changes, or update it infrequently and risk model inaccuracy and the resulting system degradation.

At least three algorithmic approaches can be followed in order to address this dilemma: The *periodic* approach is to rebuild the model from time to time. The *incremental* approach is to update the model with every change of the data. Last, the *reactive* approach, what we describe here, is to monitor the change and rebuild the model only when it no longer suits the data. The benefit of the periodic approach is its simplicity and its fixed costs in terms of communication and computation. However, the costs are fixed independent of the fact that the data is static or rapidly changing. In the former case the periodic approach wastes resources, while on the latter it might be inaccurate. The benefit of the incremental approach is that its accuracy can be optimal. Unfortunately, coming up with incremental algorithms which are both accurate and efficient can be hard and problem specific. On the other hand, model accuracy is usually judged according to a small number of rather simple metrics (misclassification error, least square error, etc.). If monitoring is done efficiently and accurately, then the reactive approach can be applied to many different data mining algorithm at low costs.

Local algorithms are one of the most efficient family of algorithms developed for distributed systems. Local algorithms are in-network algorithms in which data is never centralized but rather computation is performed by the peers of the network. At the heart of a local algorithm there is a data dependent criteria dictating when nodes can avoid sending updates to their neighbors. An algorithm is generally called local if this criteria is independent with respect to the number of nodes in the network. Therefore, in a local algorithm, it often happens that the overhead is independent of the size of the system. Primarily for this reason, local algorithms exhibit high scalability. The dependence on the criteria for avoiding to send messages also makes local algorithms inherently incremental. Specifically, if the data changes in a way that does not violate the criteria, then the algorithm adjusts to the change without sending any message.

Local algorithms were developed, in recent years, for a large selection of data modeling problems. These include association rule mining [1], facility location [2], outlier detection [3], L2 norm monitoring [4], classification [5], and multivariate regression [6]. In all these cases, resource consumption was shown to converge to a constant when the number of nodes is increased. Still, the main problem with local algorithms, thus far, has been the need to develop one for every specific problem.

In this work we make the following progress. First, we generalize a common theorem underlying the local algorithms in [1], [2], [4], [5], [6] extending it from $\mathbb{R}$ to $\mathbb{R}^d$. Next, we describe a generic algorithm, relying on the said generalized theorem, which can be used to compute arbitrarily complex functions of the average of the data in a distributed system; we show how the said algorithm can be extended to other linear combinations of data, including weighted averages of selections from the data. Then, we describe a general framework for monitoring, and consequent reactive updating of any model of horizontally distributed data. Finally, we describe the application of this framework for the problem of providing a

A preliminary version of this work was published in the Proceedings of the 2006 SIAM Data Mining Conference (SDM'06).

Manuscript received ...; revised ....

Ran Wolff is with the Department of Management Information Systems, Haifa University, Haifa-31905, Israel. Email:rwolff@mis.haifa.il. Kanishka Bhaduri is with Mission Critical Technologies Inc at NASA Ames Research Center, Moffett Field CA 94035. Email:kanishka_bh@yahoo.com. Hillol Kargupta is with the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, Baltimore, MD 21250. E-mail:hillol@cs.umbc.edu. Hillol Kargupta is also affiliated to AGNIK LLC, Columbia, MD 21045. This work was done when Kanishka Bhaduri was at UMBC.

k clustering which is a good approximation of the k-means clustering of data distributed over a large distributed system. Our theoretical and algorithmic results are accompanied with a thorough experimental validation, which demonstrates both the low cost and the excellent accuracy of our method.

The rest of this paper is organized as follows. The next section describes our notations, assumptions, and the formal problem definition. In Section III we describe and prove the main theorem of this paper. Following, Section IV describes the generic algorithm and its specification for the L2 thresholding problem. Section V presents the reactive algorithms for monitoring three typical data mining problems – *viz.* means monitoring and k-means monitoring. Experimental evaluation is presented in Section VI while Section VII describes related work. Finally, Section VIII concludes the paper and lists some prospective future work.

II. Notations, Assumptions, and Problem Definition

In this section we discuss the notations and assumptions which will be used throughout the rest of the paper. The main idea of the algorithm is to have peers accumulate sets of input vectors (or summaries thereof) from their neighbors. We show that under certain conditions on the accumulated vectors a peer can stop sending vectors to its neighbors long before it collects all input vectors. Under these conditions one of two things happens: Either all peers can compute the result from the input vectors they have already accumulated or at least one peer will continue to update its neighbors – and through them the entire network – until all peers compute the correct result.

A. Notations

Let $V = \{p_1, \dots, p_n\}$ be a set of peers (we use the term peers to describe the peers of a peer-to-peer system, motes of a wireless sensor network, etc.) connected to one another via an underlying communication infrastructure. The set of peers with which p_i can directly communicate, N_i, is known to p_i. Assuming connectedness, N_i always contains p_i and at least one more peer. Additionally, p_i is given a time varying set of input vectors in $\mathbb{R}^d$.

Peers communicate with one another by sending sets of input vectors (below, we show that for our purposes statistics on sets are sufficient). We denote by $X_{i,j}$ the latest set of vectors sent by peer p_i to p_j. For ease of notation, we denote the input of p_i (mentioned above) $X_{i,i}$. Thus, $\bigcup_{p_j \in N_i} X_{j,i}$ becomes the latest set of input vectors known to p_i.

Assuming reliable messaging, once a message is delivered both p_i and p_j know both $X_{i,j}$ and $X_{j,i}$. We further define four sets of vectors that are central to our algorithm.

Definition 2.1: The *knowledge* of p_i, is $\mathcal{K}_i = \bigcup_{p_j \in N_i} X_{j,i}$.

Definition 2.2: The *agreement* of p_i and any neighbor $p_j \in N_i$ is $\mathcal{A}_{i,j} = X_{i,j} \cup X_{j,i}$.

Definition 2.3: The *withheld knowledge* of p_i with respect to a neighbor p_j is the subtraction of the agreement from the knowledge $\mathcal{W}_{i,j} = \mathcal{K}_i \setminus \mathcal{A}_{i,j}$.

Definition 2.4: The *global input* is the set of all inputs $\mathcal{G} = \bigcup_{p_i \in V} X_{i,i}$.

We are interested in inducing functions defined on $\mathcal{G}$. Since $\mathcal{G}$ is not available at any peer, we derive conditions on $\mathcal{K}$, $\mathcal{A}$ and $\mathcal{W}$ which will allow us to learn the function on $\mathcal{G}$. Our next set of definitions deal with convex regions which are a central point of our main theorem to be discussed in the next section.

A region $R \subseteq \mathbb{R}^d$ is convex, if for every two points $x, y \in R$ and every $\alpha \in [0, 1]$, the weighted average $\alpha \cdot x + (1 - \alpha) \cdot y \in R$. Let $\mathcal{F}$ be a function from $\mathbb{R}^d$ to an arbitrary domain $\mathbb{O}$. $\mathcal{F}$ is constant on R if $\forall x, y \in R : \mathcal{F}(x) = \mathcal{F}(y)$. Any set or regions $\{R_1, R_2, \dots\}$ induces a cover of $\mathbb{R}^d$, $\mathcal{R} = \{R_1, R_2, \dots, T\}$ in which the *tie* region T includes any point of $\mathbb{R}^d$ which is not included by one of the other regions. We denote a given cover $\mathcal{R}_\mathcal{F}$ *respective* of $\mathcal{F}$ if for all regions except the tie region $\mathcal{F}$ is constant. Finally, for any $x \in \mathbb{R}^d$ we denote $\mathcal{R}_\mathcal{F}(x)$ the first region of $\mathcal{R}_\mathcal{F}$ which includes x.

B. Assumptions

Throughout this paper, we make the following assumptions:

Assumption 2.1: Communication is reliable.

Assumption 2.2: Communication takes place over a spanning communication tree.

Assumption 2.3: Peers are notified on changes in their own data x_i, and in the set of their neighbors N_i.

Assumption 2.4: Input vectors are unique.

Assumption 2.5: A respective cover $\mathcal{R}_\mathcal{F}$ can be precomputed for $\mathcal{F}$.

Note that assumption 2.1 can easily be enforced in all architectures as the algorithm poses no requirement for ordering or timeliness of messages. Simple approaches, such as piggybacking message acknowledgement can thus be implemented in even the most demanding scenarios – those of wireless sensor networks. Assumption 2.3 can be enforced using a heartbeat mechanism. Assumption 2.2 is the strongest of the three. Although solutions that enforce it exist (see for example [7]), it seems a better solution would be to remove it altogether using a method as described by Liss *et al.* [8]. However, describing such a method in this generic setting is beyond the scope of this paper. Assumption 2.4 can be enforced by adding the place and time of origin to each point and then ignoring it in the calculation of $\mathcal{F}$. Assumption 2.5 does not hold for any function. However, it does hold for many interesting ones. The algorithm described here can be sensitive to an inefficient choice of respective cover.

Note that, the correctness of the algorithm cannot be guaranteed in case the assumptions above do not hold. Specifically, duplicate counting of input vectors can occur if Assumption 2.2 does not hold — leading to any kind of result. If messages are lost then not even consensus can be guaranteed. The only positive result which can be proved quite easily is that if at any time the communication infrastructure becomes a forest, any tree will converge to the value of the function on the input of the peers belonging to that tree.

C. Sufficient statistics

The algorithm we describe in this paper deals with computing functions of linear combinations of vectors in $\mathcal{G}$. For clarity, we will focus on one such combination – the average. Linear combinations, and the average among them, can be computed from statistics. If each peer learns any input vector (other than its own) through just one of its neighbors, then for the purpose of computing $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, and $\mathcal{W}_{i,j}$, the various $X_{i,j}$ can be replaced with their average, $\overline{X_{i,j}}$, and their size, $|X_{i,j}|$. To make sure that happens, all that is required from the algorithm is that the content of every message sent by p_i to its neighbor p_j would not be dependent on messages p_j previously sent to p_i. In this way, we can rewrite:

- $|\mathcal{K}_i| = \sum_{p_j \in N_i} |X_{j,i}|$
- $|\mathcal{A}_{i,j}| = |X_{i,j}| + |X_{j,i}|$
- $|\mathcal{W}_{i,j}| = |\mathcal{K}_i| - |\mathcal{A}_{i,j}|$
- $\overline{\mathcal{K}_i} = \sum_{p_j \in N_i} \frac{|X_{j,i}|}{|\mathcal{K}_i|} \overline{X_{j,i}}$
- $\overline{\mathcal{A}_{i,j}} = \frac{|X_{i,j}|}{|\mathcal{A}_{i,j}|} \overline{X_{i,j}} + \frac{|X_{j,i}|}{|\mathcal{A}_{i,j}|} \overline{X_{j,i}}$
- $\overline{\mathcal{W}_{i,j}} = \frac{|\mathcal{K}_i|}{|\mathcal{W}_{i,j}|} \overline{\mathcal{K}_i} - \frac{|\mathcal{A}_{i,j}|}{|\mathcal{W}_{i,j}|} \overline{\mathcal{A}_{i,j}}$ or nil in case $|\mathcal{W}_{i,j}| = 0$.

D. Problem Definition

We now formally define the kind of computation provided by our generic algorithm and our notion of correct and of accurate computation.

Problem definition: Given a function $\mathcal{F}$, a spanning network tree $G(V,E)$ which might change with time, and a set of time varying input vectors $X_{i,i}$ at every $p_i \in V$, the problem is to compute the value of $\mathcal{F}$ over the average of the input vectors $\overline{\mathcal{G}}$.

While the problem definition is limited to averages of data it can be extended to weighted averages by simulation. If a certain input vector needs to be given an integer weight ω then ω peers can be simulated inside the peer that has that vector and each be given that input vector. Likewise, if it is desired that the average be taken only over those inputs which comply with some selection criteria then each peer can apply that criteria to $X_{i,i}$ apriori and then start off with the filtered data. Thus, the definition is quite conclusive.

Because the problem is defined for data which may change with time, a proper definition of algorithmic correctness must also be provided. We define the *accuracy* of an algorithm as the number of peers which compute the correct result at any given time, and denote an algorithm as *robust* if it presents constant accuracy when faced with stationarily changing data. We denote an algorithm as *eventually correct* if, once the data stops changing, and regardless of previous changes, the algorithm is guaranteed to converge to a hundred percent accuracy.

Finally, the focus of this paper is on *local* algorithms. As defined in [1], a local algorithm is one whose performance is not inherently dependent on the system size, *i.e.*, in which $|V|$ is not a factor in any lower bound on performance. Notice locality of an algorithm can be conditioned on the data. For instance, in [1] a majority voting algorithm is described which may perform as badly as $O\left(|V|^2\right)$ in case the vote is tied. Nevertheless when the vote is significant and the distribution of votes is random the algorithm will only consume constant resources, regardless of $|V|$. Alternative definitions exist for local algorithms and are thoroughly discussed in [9] and [10].

III. Main Theorems

The main theorem of this paper lay the background for a local algorithm which guarantees eventual correctness in the computation of a wide range of ordinal functions. The theorem generalizes the local stopping rule described in [1] by describing a condition which bounds the whereabouts of the global average vector in $\mathbb{R}^d$ depending on the $\mathcal{K}_i$ $\mathcal{A}_{i,j}$ and $\mathcal{W}_{i,j}$ of each peer p_i.

Theorem 3.1: **[Main Theorem]** Let $G(V,E)$ be a spanning tree in which V is a set of peers and let $X_{i,i}$ be the input of p_i, $\mathcal{K}_i$ be its knowledge, and $\mathcal{A}_{i,j}$ and $\mathcal{W}_{i,j}$ be its agreement and withheld knowledge with respect to a neighbor $p_j \in N_i$ as defined in the previous section. Let $R \subseteq \mathbb{R}^d$ be any convex region. If at a given time no messages traverse the network and for all p_i and $p_j \in N_i$ $\overline{\mathcal{K}_i}, \overline{\mathcal{A}_{i,j}} \in R$ and either $\mathcal{W}_{i,j} = \emptyset$ or $\overline{\mathcal{W}_{i,j}} \in R$ as well, then $\overline{\mathcal{G}} \in R$.

Proof: Consider a communication graph $G(V,E)$ in which for some convex R and every p_i and p_j such that $p_j \in N_i$ it holds that $\overline{\mathcal{K}_i}, \overline{\mathcal{A}_{i,j}} \in R$ and either $\mathcal{W}_{i,j} = \emptyset$ or $\overline{\mathcal{W}_{i,j}} \in R$ as well. Assume an arbitrary leaf p_i is eliminated and all of the vectors in $\mathcal{W}_{i,j}$ are added to its sole neighbor p_j. The new knowledge of p_j is $\mathcal{K}'_j = \mathcal{K}_j \cup \mathcal{W}_{i,j}$. Since by definition $\mathcal{K}_j \cap \mathcal{W}_{i,j} = \emptyset$, the average vector of the new knowledge of p_j, $\overline{\mathcal{K}'_j}$, can be rewritten as $\overline{\mathcal{K}_j \cup \mathcal{W}_{i,j}} = \alpha \cdot \overline{\mathcal{K}_j} + (1-\alpha) \cdot \overline{\mathcal{W}_{i,j}}$ for some $\alpha \in [0,1]$. Since R is convex, it follows from $\overline{\mathcal{K}_j}, \overline{\mathcal{W}_{i,j}} \in R$ that $\overline{\mathcal{K}'_j} \in R$ too.

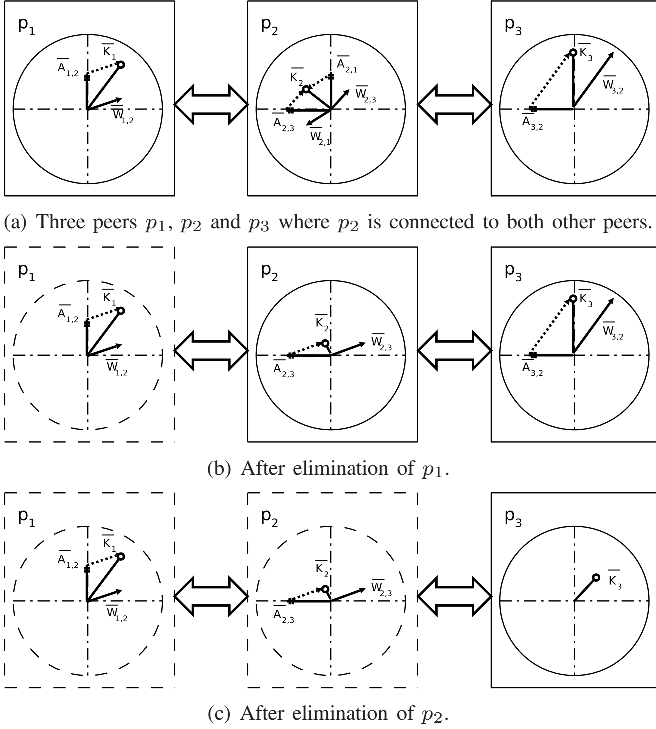
Now, consider the change in the withheld knowledge of p_j with respect to any other neighbor $p_k \in N_j$ resulting from sending such a message. The new $\mathcal{W}'_{j,k} = \mathcal{W}_{i,j} \cup \mathcal{W}_{j,k}$. Again, since $\mathcal{W}_{i,j} \cap \mathcal{W}_{j,k} = \emptyset$ and since R is convex it follows from $\mathcal{W}_{i,j}, \mathcal{W}_{j,k} \in R$ that $\mathcal{W}'_{j,k} \in R$ as well. Finally, notice the agreements of p_j with any neighbor p_k except p_i do not change as a result of such message.

Hence, following elimination of p_i we have a communication tree with one less peer in which the same conditions still apply to every remaining peer and its neighbors. Proceeding with elimination we can reach a tree with just one peer p_1, still assured that $\overline{\mathcal{K}_1} \in R$. Moreover, since no input vector was lost at any step of the elimination $\mathcal{K}_1 = \mathcal{G}$. Thus, we have that under the said conditions $\overline{\mathcal{G}} \in R$. ■

Theorem 3.1 is exemplified in Figure 1. Three peers are shown, each with a drawing of its knowledge, it agreement with its neighbor or neighbors, and the withheld knowledge. Notice the agreement $\overline{\mathcal{A}_{1,2}}$ drawn for p_1 is identical to $\overline{\mathcal{A}_{2,1}}$ at p_2. For graphical simplicity we assume all of the vectors have the same weight – and avoid expressing it. We also depict the withheld knowledge vectors twice – once as a subtraction of the agreement from the knowledge – using a dotted line – and once – shifted to the root – as measured in practice. If the position of the three peers' data is considered vis-a-vis the circular region then the conditions of Theorem 3.1 hold.

Now, assume what would happen when peer p_1 is eliminated. This would mean that all of the knowledge it withholds from p_2 is added to $\mathcal{K}_2$ and to $\mathcal{W}_{2,3}$. Since we assumed $|\mathcal{W}_{1,2}| = |\mathcal{K}_2| = 1$ the result is simply the averaging of the previous $\overline{\mathcal{K}_2}$ and $\overline{\mathcal{W}_{1,2}}$. Notice both these vectors remain in the circular region.

Lastly, as p_2 is eliminated as well, $\mathcal{W}_{2,3}$ – which now also includes $\mathcal{W}_{1,2}$ – is blended into the knowledge of p_3. Thus, $\mathcal{K}_3$ becomes equal to $\mathcal{G}$. However, the same argument, as applied in the elimination of p_1, assures the new $\overline{\mathcal{K}_3}$ is in the circular region as well.



(a) Three peers p_1, p_2 and p_3 where p_2 is connected to both other peers.

(b) After elimination of p_1.

(c) After elimination of p_2.

Fig. 1. At Figure 1(a) the data at all three peers concur with the conditions of Theorem 3.1 with respect to the circle – which is a convex region. If subsequently peer p_1 is eliminated and $\mathcal{W}_{1,2}$ sent to p_2 then $\mathcal{A}_{2,3}$ is not affected and $\mathcal{K}_2$ and $\mathcal{W}_{2,3}$ do change but still remain in the same region. When subsequently, in Figure 1(c), p_2 is eliminated again $\mathcal{K}_3 = \mathcal{G}$ which demonstrates $\overline{\mathcal{G}}$ is in the circular region.

To see the relation of Theorem 3.1 to the previous the Majority-Rule algorithm [1], one can restate the majority voting problem as deciding whether the average of zero-one votes is in the segment $[0, \lambda)$ or the segment $[\lambda, 1]$. Both segments are convex, and the algorithm only stops if for all peers the knowledge is further away from λ than the agreement – which is another way to say the knowledge, the agreement, and the withheld data are all in the same convex region. Therefore, Theorem 3.1 generalizes the basic stopping rule of Majority-Rule to any convex region in $\mathbb{R}^d$.

Two more issues arise from this comparison: one is that in Majority-Rule the regions used by the stopping rule coincide with the regions in which $\mathcal{F}$ is constant. The other is that in the Majority-Rule, every peer decides according to which of the two regions it should try to stop by choosing the region which includes the agreement. Since there are just two non-overlapping region, peers reach consensus on the choice of region and, hence, on the output.

These two issues become more complex for a general $\mathcal{F}$ over $\mathbb{R}^d$. First, for many interesting $\mathcal{F}$, the regions in which the function is constant are not all convex. Also, there could be many more than two such regions, and the selection of the region in which the stopping rule needs be evaluated becomes non-trivial.

We therefore provide two lemmas which provide a way to deal with the selection problem and an answer to the case where in which a function cannot be neatly described as a partitioning of $\mathbb{R}^d$ to convex regions in which it is constant.

Lemma 3.2: **[Consensus]** Let $G(V, E)$ be a spanning tree in which V is a set of peers and let $X_{i,i}$ be the input of p_i, $\mathcal{K}_i$ be its knowledge, and $\mathcal{A}_{i,j}$ and $\mathcal{W}_{i,j}$ be its agreement and withheld knowledge with respect to a neighbor $p_j \in N_i$ as defined in the previous section. Let $\mathcal{R_F} = \{R_1, R_2, \ldots, T\}$ be a $\mathcal{F}$-respective cover, and let $\mathcal{R_F}(x)$ be the first region in $\mathcal{R_F}$ which contains x. If for every peer p_i and every $p_j \in N_i$ $\mathcal{R_F}\left(\overline{\mathcal{K}_i}\right) = \mathcal{R_F}\left(\overline{\mathcal{A}_{i,j}}\right)$ then for every two peers p_i and p_ℓ, $\mathcal{R_F}\left(\overline{\mathcal{K}_i}\right) = \mathcal{R_F}\left(\overline{\mathcal{K}_\ell}\right)$.

Proof: We prove this by contradiction. Assume that the result is not true. Then there are two peers p_i and p_ℓ with $\mathcal{R_F}\left(\overline{\mathcal{K}_i}\right) \neq \mathcal{R_F}\left(\overline{\mathcal{K}_\ell}\right)$. Since the communication graph is a spanning tree, there is a path from p_i to p_ℓ and somewhere along that path there are two neighbor peers, p_u and p_v such that $\mathcal{R_F}\left(\overline{\mathcal{K}_u}\right) \neq \mathcal{R_F}\left(\overline{\mathcal{K}_v}\right)$. Notice, however, that $\overline{\mathcal{A}_{u,v}} = \overline{\mathcal{A}_{v,u}}$. Therefore, either $\mathcal{R_F}\left(\overline{\mathcal{K}_u}\right) \neq \mathcal{R_F}\left(\overline{\mathcal{A}_{u,v}}\right)$ or $\mathcal{R_F}\left(\overline{\mathcal{K}_v}\right) \neq \mathcal{R_F}\left(\overline{\mathcal{A}_{v,u}}\right)$ — a contradiction. ■

Building on Lemma 3.2 above, a variant of Theorem 3.1 can be proved which makes use of a respective cover to compute the value of $\mathcal{F}$.

Theorem 3.3: Let $G(V, E)$ be a spanning tree in which V is a set of peers and let $X_{i,i}$ be the input of p_i, $\mathcal{K}_i$ be its knowledge, and $\mathcal{A}_{i,j}$ and $\mathcal{W}_{i,j}$ be its agreement and withheld knowledge with respect to a neighbor $p_j \in N_i$ as defined in the previous section. Let $\mathcal{R_F} = \{R_1, R_2, \ldots, T\}$ be a respective cover, and let $\mathcal{R_F}(x)$ be the first region in $\mathcal{R_F}$ which contains x. If for every peer p_i and every $p_j \in N_i$ $\mathcal{R_F}\left(\overline{\mathcal{K}_i}\right) = \mathcal{R_F}\left(\overline{\mathcal{A}_{i,j}}\right) \neq T$ and if furthermore either $\mathcal{W}_{i,j} = \emptyset$ or $\overline{\mathcal{W}_{i,j}} \in \mathcal{R_F}\left(\overline{\mathcal{K}_i}\right)$ then for every p_i, $\mathcal{F}(\overline{\mathcal{K}_i}) = \mathcal{F}(\overline{\mathcal{G}})$.

Proof: From Lemma 3.2 it follows that all peers compute the same $\mathcal{R_F}\left(\overline{\mathcal{K}_i}\right)$. Thus, since this region is not T, it must be convex. It therefore follows from Theorem 3.1 that $\overline{\mathcal{G}}$ is, too, in $\mathcal{R_F}\left(\overline{\mathcal{K}_i}\right)$. Lastly, since $\mathcal{R_F}$ is a respective cover $\mathcal{F}$ must be constant on all regions except T. Thus, the value of $\mathcal{F}(\overline{\mathcal{G}})$ is equal to that of $\mathcal{F}(\overline{\mathcal{K}_i})$, for any p_i. ■

IV. A Generic Algorithm and Its Instantiation

This section describes a generic algorithm which relies on the results presented in the previous section to compute the value of a given function of the average of the input vectors. This generic algorithm is both local and eventually correct. The section proceeds to exemplify how this generic algorithm can be used by instantiating it to compute whether the average vector has length above a given threshold $\mathcal{F}(x) =$

$\begin{cases}0 & \|x\| \leq \epsilon \\ 1 & \|x\| > \epsilon\end{cases}$. L2 thresholding is both an important problem in its own right and can also serve as the basis for data mining algorithms as will be described in the next section.

A. Generic Algorithm

The generic algorithm, depicted in Algorithm 1, receives as input the function $\mathcal{F}$, a respective cover $\mathcal{R}_{\mathcal{F}}$, and a constant, L, whose function is explained below. Each peer p_i outputs, at every given time, the value of $\mathcal{F}$ based on its knowledge $\overline{\mathcal{K}_i}$.

The algorithm is event driven. Events could be one of the following: a message from a neighbor peer, a change in the set of neighbors (*e.g.*, due to failure or recovery), a change in the local data, or the expiry of a timer which is always set to no more than L. On any such event p_i calls the **OnChange** method. When the event is a message $\overline{X}, |X|$ received from a neighbor p_j, p_i would update $\overline{X_{i,j}}$ to $\overline{X}$ and $|X_{i,j}|$ to $|X|$ before it calls **OnChange**.

The objective of the **OnChange** method is to make certain that the conditions of Lemma 3.3 are maintained for the peer that runs it. These conditions require $\overline{\mathcal{K}_i}$, $\overline{\mathcal{A}_{i,j}}$, and $\overline{\mathcal{W}_{i,j}}$ (in case it is not null) to all be in $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$, which is not the tie region T. Of the three, $\overline{\mathcal{K}_i}$ cannot be manipulated by the peer. The peer thus manipulates both $\overline{\mathcal{A}_{i,j}}$, and $\overline{\mathcal{W}_{i,j}}$ by sending a message to p_j, and subsequently updating $X_{i,j}$.

In case $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right) \neq T$ one way to adjust $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$ so that the conditions of Lemma 3.3 are maintained is to send the entire $\mathcal{W}_{i,j}$ to p_j. This would make $\mathcal{A}_{i,j}$ equal to $\mathcal{K}_i$, and therefore make $\overline{\mathcal{A}_{i,j}}$ equal to $\overline{\mathcal{K}_i}$ and in $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$. Additionally, $\mathcal{W}_{i,j}$ becomes empty. However, this solution is one of the many possible changes to $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$, and not necessarily the optimal one. We leave the method of finding a value for the next message $X_{i,j}$ which should be sent by p_i unspecified at this stage, as it may depend on characteristics of the specific $\mathcal{R}_{\mathcal{F}}$.

The other possible case is that $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right) = T$. Since T is always the last region of $\mathcal{R}_{\mathcal{F}}$, this means $\overline{\mathcal{K}_i}$ is outside any other region $R \in \mathcal{R}_{\mathcal{F}}$. Since T is not necessarily convex, the only option which will guarantee eventual correctness in this case is if p_i sends the entire withheld knowledge to every neighbor it has.

Lastly, we need to address the possibility that although $|\mathcal{W}_{i,j}| = 0$ we will have $\overline{\mathcal{A}_{i,j}}$ which is different from $\overline{\mathcal{K}_i}$. This can happen, *e.g.*, when the withheld knowledge is sent in its entirety and subsequently the local data changes. Notice this possibility results only from our choice to use sufficient statistics rather than sets of vectors: Had we used sets of vectors, $\mathcal{W}_{i,j}$ would not have been empty, and would fall into one of the two cases above. As it stands, we interpret the case of non-empty $\mathcal{W}_{i,j}$ with zero $|\mathcal{W}_{i,j}|$ as if $\overline{\mathcal{W}_{i,j}}$ is in T.

It should be stressed here that if the conditions of Lemma 3.3 hold the peer does not need to do anything even if its knowledge changes. The peer can rely on the correctness of the general results from the previous section which assure that if $\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$ is not the correct answer then eventually one of its neighbors will send it new data and change $\overline{\mathcal{K}_i}$. If, one the other hand, one of the aforementioned cases do occur, then p_i sends a message. This is performed by the **SendMessage** method. If $\overline{\mathcal{K}_i}$ is in T then p_i simply sends all of the withheld data. Otherwise, a message is computed which will assure $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$ are in $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$.

One last mechanism employed in the algorithm is a "leaky bucket" mechanism. This mechanism makes certain no two messages are sent in a period shorter than a constant L. Leaky bucket is often used in asynchronous, event-based systems to prevent event inflation. Every time a message needs to be sent, the algorithm checks how long has it been since the last one was sent. If that time is less than L, the algorithm sets a timer for the reminder of the period and calls **OnChange** again when the timer expires. Note that this mechanism does not enforce any kind of synchronization on the system. It also does not affect correctness: at most it can delay convergence because information would propagate more slowly.

Algorithm 1 Generic Local Algorithm

Input of peer p_i**:** $\mathcal{F}$, $\mathcal{R}_{\mathcal{F}} = \{R_1, R_2, \ldots, T\}$, L, $X_{i,i}$, and N_i
Ad hoc output of peer p_i**:** $\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$
Data structure for p_i**:** For each $p_j \in N_i$ $\overline{X_{i,j}}$, $|X_{i,j}|$, $\overline{X_{j,i}}$, $|X_{i,j}|$, $last_message$
Initialization: $last_message \leftarrow -\infty$
On receiving a message $\overline{X}, |X|$ **from** p_j**:**
– $\overline{X_{j,i}} \leftarrow \overline{X}$, $|X_{j,i}| \leftarrow |X|$
On change in $X_{i,i}$, N_i, $\overline{\mathcal{K}_i}$ or $|\mathcal{K}_i|$: call OnChange()
OnChange()
For each $p_j \in N_i$:
– If one of the following conditions occur:
– 1. $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right) = T$ and either $\overline{\mathcal{A}_{i,j}} \neq \overline{\mathcal{K}_i}$ or $|\mathcal{A}_{i,j}| \neq |\mathcal{K}_i|$
– 2. $|\mathcal{W}_{i,j}| = 0$ and $\overline{\mathcal{A}_{i,j}} \neq \overline{\mathcal{K}_i}$
– 3. $\mathcal{A}_{i,j} \notin \mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$ or $\mathcal{W}_{i,j} \notin \mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$
– then
– – call SendMessage(p_j)
SendMessage(p_j)**:**
If $time\left(\right) - last_message \geq L$
– If $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right) = T$ then the new $\overline{X_{i,j}}$ and $|X_{i,j}|$ are $\overline{\mathcal{W}_{i,j}}$ and $|\mathcal{W}_{i,j}|$, respectively
– Otherwise compute new $\overline{X_{i,j}}$ and $|X_{i,j}|$ such that $\overline{\mathcal{A}_{i,j}} \in \mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$ and either $\overline{\mathcal{W}_{i,j}} \in \mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$ or $|\mathcal{W}_{i,j}| = 0$
– $last_message \leftarrow time\left(\right)$
– Send $\overline{X_{i,j}}, |X_{i,j}|$ to p_j
Else
– Wait $L - \left(time\left(\right) - last_message\right)$ time units and then call OnChange()

B. Eventual correctness

Proving eventual correctness requires showing that if both the underlying communication graph and the data at every peer cease to change then after some length of time every peer would output the correct result $\mathcal{F}\left(\overline{\mathcal{G}}\right)$; and that this would happen for *any* static communication tree $G(V,E)$, *any* static data $X_{i,i}$ at the peers, and any possible state of the peers.

Proof: [**Eventual Correctness**] Regardless of the state of $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, $\mathcal{W}_{i,j}$, the algorithm will continue to send messages, and accumulate more and more of $\mathcal{G}$ in each $\mathcal{K}_i$ until one of two things happens: One is that for every peer $\mathcal{K}_i = \mathcal{G}$ and thus $\mathcal{A}_{i,j} = \mathcal{K}_i$ for all $p_j \in N_i$. Alternatively, for every p_i $\overline{\mathcal{A}_{i,j}}$ is in $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$, which is different than T, and $\mathcal{W}_{i,j}$ is either in $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$ as well or is empty. In the former case, $\overline{\mathcal{K}_i} = \overline{\mathcal{G}}$, so every peer obviously computes $\mathcal{F}\left(\overline{\mathcal{K}_i}\right) = \mathcal{F}\left(\overline{\mathcal{G}}\right)$. In the latter case, Theorem 3.1 dictates that $\overline{\mathcal{G}} \in R_\ell$, so $\mathcal{F}\left(\overline{\mathcal{K}_i}\right) = \mathcal{F}\left(\overline{\mathcal{G}}\right)$ too. Finally, provided that every message sent in the algorithm carries the information of at least one input vector to a peer that still does not have it, the number of messages sent between the time the data stops changing and the time in which every peer has the data of all other peers is bounded by $O\left(|V|^2\right)$. ■

C. Local L2 Norm Thresholding

Following the description of a generic algorithm, specific algorithms can be implemented for various functions $\mathcal{F}$. One of the most interesting functions (also dealt with in our previous paper [4]) is that of thresholding the L2 norm of the average vector, i.e., deciding if $\left\|\overline{\mathcal{G}}\right\| \leq \epsilon$.

To produce a specific algorithm from the generic one, the following two steps need to be taken:

1) A respective cover $\mathcal{R}_{\mathcal{F}}$, needs to be found
2) A method for finding $\overline{X_{i,j}}$ and $|X_{i,j}|$ which assures that both $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$ are in R needs to be formulated

In the case of L2 thresholding, the area for which $\mathcal{F}$ outputs $true$ – the inside of an ϵ circle – is convex. This area is denoted R_{in}. The area outside the ϵ-circle can be divided by randomly selecting unit vectors $\hat{u_1}, \ldots, \hat{u_\ell}$ and then drawing the half-spaces $H_j = \{\vec{x} : \vec{x} \cdot \hat{u_j} \geq \epsilon\}$. Each half-space is convex. Also, they are entirely outside the ϵ circle, so $\mathcal{F}$ is constant on every H_j. $\{R_{in}, H_1, \ldots, H_\ell, T\}$ is, thus, a respective cover. Furthermore, by increasing ℓ, the area between the halfspaces and the circle or the tie area can be minimized to any desired degree.

It is left to describe how the **SendMessage** method computes a message that forces $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$ into the region which contains $\overline{\mathcal{K}_i}$ if they are not in it. A related algorithm, Majority-Rule [1], suggests sending all of the withheld knowledge in any case. However, experiments with dynamic data hint this method may be unfavorable. If all or most of the knowledge is sent and the data later changes the withheld knowledge becomes the difference between the old and the new data. This difference tends to be far more noisy than the original data. Thus, while the algorithm makes certain $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$ are brought into the same region as $\overline{\mathcal{K}_i}$, it still makes an effort to maintain some withheld knowledge.

Although it may be possible to optimize the size of $|\mathcal{W}_{i,j}|$ we take the simple and effective approach of testing an exponentially decreasing sequence of $|\mathcal{W}_{i,j}|$ values, and then choosing the first such value satisfying the requirements for $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$. When a peer p_i needs to send a message, it first sets the new $\overline{X_{i,j}}$ to $\frac{|\mathcal{K}_i|\overline{\mathcal{K}_i} - |X_{j,i}|\overline{X_{j,i}}}{|\mathcal{K}_i| - |X_{j,i}|}$. Then, it tests a sequence of values for $|X_{i,j}|$. Clearly, $|X_{i,j}| = |\mathcal{K}_i| - |X_{j,i}|$ translates to an empty withheld knowledge and must concur with the conditions of Lemma 3.3. However, the algorithm begins with $|X_{i,j}| = \frac{|\mathcal{K}_i| - |X_{j,i}|}{2}$ and only gradually increases the weight, trying to satisfy the conditions without sending all data.

Algorithm 2 Local L2 Thresholding

Input of peer p_i**:** ϵ, L, $X_{i,i}$, N_i, ℓ
Global constants: A random seed s
Data structure for p_i**:** For each $p_j \in N_i$ $\overline{X_{i,j}}$, $|X_{i,j}|$, $\overline{X_{j,i}}$, $|X_{i,j}|$, $last_message$
Output of peer p_i**:** 0 if $\left\|\overline{\mathcal{K}_i}\right\| \leq \epsilon$, 1 otherwise
Computation of $\mathcal{R}_{\mathcal{F}}$**:**
Let $R_{in} = \{\vec{x} : \|\vec{x}\| \leq \epsilon\}$
Let $\hat{u}_1, \ldots, \hat{u}_\ell$ be pseudo-random unit vectors and let $H_j = \{\vec{x} : \vec{x} \cdot \hat{u}_j \geq \epsilon\}$
$\mathcal{R}_{\mathcal{F}} = \{R_{in}, H_1, \ldots, H_\ell, T\}$.
Computation of $|X_{i,j}|$ **and** $\overline{X_{i,j}}$**:**
$\overline{X_{i,j}} \leftarrow \frac{|\mathcal{K}_i|\overline{\mathcal{K}_i} - |X_{j,i}|\overline{X_{j,i}}}{|\mathcal{K}_i| - |X_{j,i}|}$
$w \leftarrow |X| \leftarrow |\mathcal{K}_i| - |X_{j,i}|$
Do
– $w \leftarrow \lfloor \frac{w}{2} \rfloor$
– $|X_{i,j}| \leftarrow |\mathcal{K}_i| - |X_{j,i}| - w$
While ($\overline{\mathcal{A}_{i,j}} \notin \mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$ or $\overline{\mathcal{W}_{i,j}} \notin \mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$ and $|\mathcal{W}_{i,j}| \neq 0$)
Initialization: $last_message \leftarrow -\infty$, compute $\mathcal{R}_{\mathcal{F}}$
On receiving a message $\overline{X}, |X|$ **from** p_j**:**
– $\overline{X_{j,i}} \leftarrow \overline{X}$, $|X_{j,i}| \leftarrow |X|$
On change in $X_{i,i}$, N_i, $\overline{\mathcal{K}_i}$ or $|\mathcal{K}_i|$: call OnChange()
OnChange()
For each $p_j \in N_i$:
– If one of the following conditions occur:
– 1. $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right) = T$ and either $\overline{\mathcal{A}_{i,j}} \neq \overline{\mathcal{K}_i}$ or $|\mathcal{A}_{i,j}| \neq |\mathcal{K}_i|$
– 2. $|\mathcal{W}_{i,j}| = 0$ and $\overline{\mathcal{A}_{i,j}} \neq \overline{\mathcal{K}_i}$
– 3. $\mathcal{A}_{i,j} \notin \mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$ or $\mathcal{W}_{i,j} \notin \mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right)$
– then
– – call SendMessage(p_j)
SendMessage(p_j)**:**
If $time\left(\right) - last_message \geq L$
– If $\mathcal{R}_{\mathcal{F}}\left(\overline{\mathcal{K}_i}\right) = T$ then the new $\overline{X_{i,j}}$ and $|X_{i,j}|$ are $\overline{\mathcal{W}_{i,j}}$ and $|\mathcal{W}_{i,j}|$, respectively
– Otherwise compute new $\overline{X_{i,j}}$ and $|X_{i,j}|$
– $last_message \leftarrow time\left(\right)$
– Send $\overline{X_{i,j}}, |X_{i,j}|$ to p_j
Else
– Wait $L - \left(time\left(\right) - last_message\right)$ time units and then call OnChange()

V. Reactive Algorithms

The previous section described an efficient generic local algorithm, capable of computing any function even when the data and system are constantly changing. In this section, we leverage this powerful tool to create a framework for producing and maintaining various data mining models. This framework is simpler than the current methodology of inventing a specific distributed algorithm for each problem and may be as efficient as its counterparts.

The basic idea of the framework is to employ a simple, costly, and possibly inaccurate *convergecast* algorithm in which a single peer samples data from the network and then computes, based on this "best-effort" sample, a data mining model. Then, this model is *broadcast* to the entire network; again, a technique which might be costly. Once, every peer is informed with the current model, a local algorithm, which is an instantiation of the generic algorithm is used in order to monitor the quality of the model. If the model is not sufficiently accurate or the data has changed to the degree that the model no longer describes it, the monitoring algorithm alerts and triggers another cycle of data collection. It is also possible to tune the algorithm by increasing the sample size if the alerts are frequent and decreasing it when they are infrequent. Since the monitoring algorithm is eventually correct, eventual convergence to a sufficiently accurate model is very likely. Furthermore, when the data only goes through stationary changes, the monitoring algorithm triggers false alerts infrequently and hence can be extremely efficient. Thus, the overall cost of the framework is low.

We describe two instantiations of this basic framework, each highlighting a different aspect. First we discuss the problem of computing the mean input vector, to a desired degree of accuracy. Then, we present an algorithm for computing a variant of the k-means clusters suitable for dynamic data.

A. Mean Monitoring

The problem of monitoring the mean of the input vectors has direct applications to many data analysis tasks. The objective in this problem is to compute a vector $\overline{\mu}$ which is a good approximation for $\overline{\mathcal{G}}$. Formally, we require that $\left\|\overline{\mathcal{G}}-\overline{\mu}\right\| \leq \epsilon$ for a desired value of ϵ.

For any given estimate $\overline{\mu}$, monitoring whether $\left\|\overline{\mathcal{G}}-\overline{\mu}\right\| \leq \epsilon$ is possible via direct application of the L2 thresholding algorithm from Section IV-C. Every peer p_i subtracts $\overline{\mu}$ from every input vector in $X_{i,i}$. Then, the peers jointly execute L2 Norm Thresholding over the modified data. If the resulting average is inside the ϵ-circle then $\overline{\mu}$ is a sufficiently accurate approximation of $\overline{\mathcal{G}}$; otherwise, it is not.

The basic idea of the mean monitoring algorithm is to employ a convergecast-broadcast process in which the convergecast part computes the average of the input vectors and the broadcast part delivers the new average to all the peers. The trick is that, before a peer sends the data it collected up the convergecast tree, it waits for an indication that the current $\overline{\mu}$ is not a good approximation of the current data. Thus, when the current $\overline{\mu}$ is a good approximation, convergecast is slow and only progresses as a result of false alerts. During this time, the cost of the convergecast process is negligible compared to that of the L2 thresholding algorithm. When, on the other hand, the data does change, all peers alert almost immediately. Thus, convergecast progresses very fast, reaches the root, and initiates the broadcast phase. Hence, a new $\overline{\mu}$ is delivered to every peer, which is a more updated estimate of $\overline{\mathcal{G}}$.

The details of the mean monitoring algorithm are given in Algorithm 3. One detail is that of an alert mitigation constant, τ, selected by the user. The idea here is that an alert should persist for a given period of time before the convergecast advances. Experimental evidence suggests that setting τ to even a fraction of the average edge delay greatly reduces the number of convergecasts without incurring a significant delay in the updating of $\overline{\mu}$.

A second detail is the separation of the data used for alerting – the input of the L2 thresholding algorithm – from that which is used for computing the new average. If the two are the same then the new average may be biased. This is because an alert, and consequently an advancement in the convergecast, is bound to be more frequent when the local data is extreme. Thus, the initial data, and later every new data, is randomly associated with one of two buffers: R_i, which is used by the L2 Thresholding algorithm, and T_i, on whom the average is computed when convergecast advances.

A third detail is the implementation of the convergecast process. First, every peer tracks changes in the knowledge of the underlying L2 thresholding algorithm. When it moves from inside the ϵ-circle to outside the ϵ-circle the peer takes note of the time, and sets a timer to τ time units. When a timer expires or when a data message is received from one of its neighbors p_i checks if currently there is an alert and if it was recorded τ or more time units ago. If so, it counts the number of its neighbors from whom it received a data message. If it received data messages from all of its neighbors, the peer moves to the broadcast phase, computes the average of its own data and of the received data and sends it to itself. If it has received data messages from all but one of the neighbors then this one neighbor becomes the peer's parent in the convergecast tree; the peer computes the average of its own and its other neighbors' data, and sends the average with its cumulative weight to the parent. Then, it moves to the broadcast phase. If two or more of its neighbors have not yet sent a data messages p_i keeps waiting.

Lastly, the broadcast phase is fairly straightforward. Every peer which receives the new $\overline{\mu}$ vector, updates its data by subtracting it from every vector in R_i and transfers those vectors to the underlying L2 thresholding algorithm. Then, it re-initializes the buffers for the data messages and sends the new $\overline{\mu}$ vector to its other neighbors and changes the status to convergecast. There could be one situation in which a peer receives a new $\overline{\mu}$ vector even though it is already in the convergecast phase. This happens when two neighbor peers concurrently become roots of the convergecast tree (i.e., when each of them concurrently sends the last convergecast message to the other). To break the tie, a root peer p_i which receives $\overline{\mu}$ from a neighbor p_j while in the convergecast phase ignores the message if $i > j$ it ignores the message. Otherwise if $i < j$ p_i treats the message just as it would in the broadcast phase.

B. k-Means Monitoring

We now turn to a more complex problem, that of computing the k-means of distributed data. The classic formulation of the k-means algorithm is a two step recursive process in which every data point is first associated with the nearest of k centroids, and then every centroid is moved to the average of the points associated with it – until the average is the same

Algorithm 3 Mean Monitoring

Input of peer p_i**:** ϵ, L, $X_{i,i}$, the set of neighbors N_i, an initial vector $\overline{\mu_0}$, an alert mitigation constant τ.
Output available to every peer p_i**:** An approximated means vector $\overline{\mu}$
Data structure of peer p_i**:** Two sets of vectors R_i and T_i, a timestamp $last_change$, flags: $alert$, $root$, and $phase$, for each $p_j \in N_i$, a vector $\overline{v}_j$ and a counter c_j
Initialization:
Set $\overline{\mu} \leftarrow \overline{\mu_0}$, $alert \leftarrow false$, $phase \leftarrow convergecast$
Split $X_{i,i}$ evenly between R_i and T_i
Initialize an L2 thresholding algorithm with the input ϵ, L, $\{\overline{x} - \overline{\mu} : \overline{x} \in R_i\}$, N_i
Set $\overline{v_i}, c_i$ to $\overline{T_i}, |T_i|$, respectively, and $\overline{v_j}, c_j$ to $\overline{0}, 0$ for all other $p_j \in N_i$
On addition of a new vector $\overline{x}$ **to** $X_{i,i}$**:**
Randomly add $\overline{x}$ to either R_i or T_i
If $\overline{x}$ was added to R_i, update the input of the L2 thresholding algorithm to $\{\overline{x} - \overline{\mu} : \overline{x} \in R_i\}$
Otherwise, update v_i and c_i.
On change in $\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$ **of the L2 thresholding algorithm:**
If $\left\|\overline{\mathcal{K}_i}\right\| \geq \epsilon$ and $alert = false$ then
– set $last_change \leftarrow time()$
– set $alert \leftarrow true$
– set a timer to τ time units
If $\left\|\overline{\mathcal{K}_i}\right\| < \epsilon$ then
– Set $alert \leftarrow false$
On receiving a data message $\overline{v}, c$ **from** $p_j \in N_i$**:**
Set $\overline{v_j} \leftarrow \overline{v}$, $c_j \leftarrow c$
Call Convergecast
On timer expiry or call to Convergecast:
If $alert = false$ return
If $time() - last_change < \tau$ set timer to $time() + \tau - last_change$ and return
If for all $p_k \in N_i$ except for one $c_k \neq 0$
– Let $s = \sum_{p_j \in N_i} c_j$, $\overline{s} = \sum_{p_j \in N_i} \frac{c_j}{s} \overline{v_j}$
– Send $s, \overline{s}$ to p_l
– Set $phase \leftarrow Broadcast$
If for all $p_k \in N_i$ $c_k \neq 0$
– Let $s = \sum_{p_j \in N_i} c_j$, $\overline{s} = \sum_{p_j \in N_i} \frac{c_j}{s} \overline{v_j}$
– Set $phase \leftarrow Convergecast$
– Send $\overline{\mu}$ to all $p_k \in N_i$
On receiving $\overline{\mu'}$ **from** $p_j \in N_i$**:**
If $phase = convergecast$ and $i > j$ then return
Set $\overline{\mu} \leftarrow \overline{\mu'}$
Replace the input of the L2 thresholding algorithm with $\{\overline{x} - \overline{\mu} : \overline{x} \in R_i\}$
Set $phase \leftarrow convergecast$ and set all c_j to 0
Send $\overline{\mu}$ to all $p_k \neq p_j \in N_i$
Other than that follow the L2 thresholding algorithm

as the centroid. To make the algorithm suitable for a dynamic data setup, we relax the stopping criteria. In our formulation, a solution is considered admissible when the average of point is within an ϵ-distance of the centroid with whom they are associated.

Similar to the mean monitoring, the k-means monitoring algorithm (Algorithm. 4) is performed in a cycle of convergecast and broadcast. The algorithm, however, is different in some important respects. First, instead of taking part of just one execution of L2 thresholding, each peer takes part in k such executions – one per centroid. The input of the ℓth execution are those points in the local data set $X_{i,i}$ for which the ℓth centroid, $\overline{c_\ell}$, is the closest. Thus, each execution monitors whether one of the centroids needs to be updated. If even one execution discovers that the norm of the respective knowledge $\left\|\overline{\mathcal{K}_i^\ell}\right\|$ is greater than ϵ, the peer alerts, and if the alert persists for τ time units the peer advances the convergecast process.

Another difference between k-means monitoring and mean monitoring is the statistics collected during convergecast. In k-means monitoring, that statistics is a sample of size b (dictated by the user) from the data. Each peer samples with returns from the samples it received from its neighbors, and from its own data, such that the probability of sampling a point is proportional to a weight. The result of this procedure is that every input point stands an equal chance to be included in the sample that arrives to the root. The root then computes the k-means on the sample, and sends the new centroids in a broadcast message.

VI. Experimental Validation

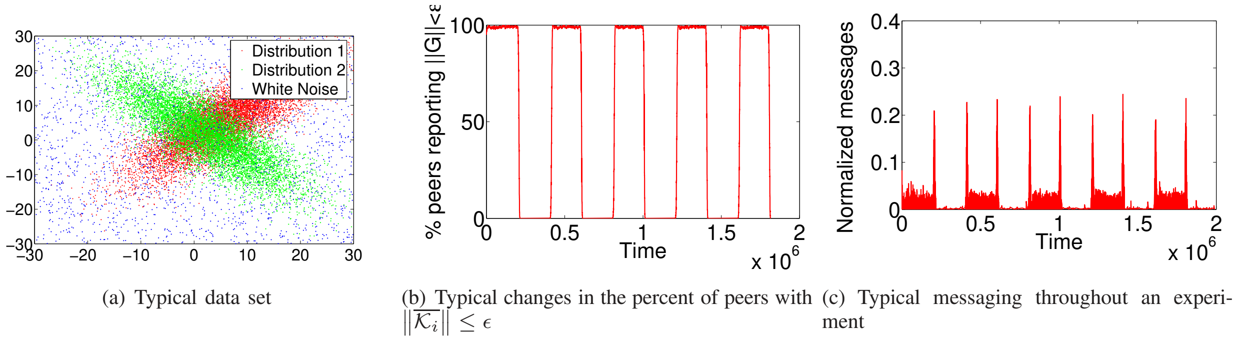
To validate the performance of our algorithms we conducted experiments on a simulated network of thousands of peers. In this section we discuss the experimental setup and analyze the performance of the algorithms.

A. Experimental Setup

Our implementation makes use of the Distributed Data Mining Toolkit (DDMT)[1]– a distributed data mining development environment from DIADIC research lab at UMBC. DDMT uses topological information which can be generate by BRITE[2], a universal topology generator from Boston University. In our simulations we used topologies generated according to the *Barabasi Albert (BA)* model, which is often considered a reasonable model for the Internet. BA also defines delays for network edges, which are the basis for our time measurement[3]. On top of the network generated by BRITE, we overlayed a spanning tree.

The data used in the simulations was generated using a mixture of Gaussians in $\mathbb{R}^d$. Every time a simulated peer needed an additional data point, it sampled d Gaussians and multiplied the resulting vector with a $d \times d$ covariance matrix in which the diagonal elements were all 1.0's while the off-diagonal elements were chosen uniformly between 1.0 and

[1] http://www.umbc.edu/ddm/wiki/software/DDMT
[2] http://www.cs.bu.edu/brite/
[3] Wall time is meaningless when simulating thousands of computers on a single PC.



(a) Typical data set

(b) Typical changes in the percent of peers with $\left\|\overline{\mathcal{K}_i}\right\| \leq \epsilon$

(c) Typical messaging throughout an experiment

Fig. 2. A typical experiment is run for 10 equal length epochs. The epochs have very similar means, and very large variance. Quality and overall cost are measured across the entire experiment – including transitional phases.

2.0. Alternatively, 10% of the points were chosen uniformly at random in the range of $\mu \pm 3\sigma$. At controlled intervals, the means of the Gaussians were changed, thereby creating an epoch change. A typical data in two dimensions can be seen in Figure 2(a). We preferred synthetic data because of the large number of factors (twelve, in our analysis) which influence the behavior of an algorithm, and the desire to perform a tightly controlled experiment in order to understand the behavior of a complex algorithm which operates in an equally as complex environment.

The two most important qualities measured in our experiments are the *quality* of the result and the *cost* of the algorithm. Quality is defined differently for the L2 thresholding algorithm, the mean monitoring algorithm, and the k-means algorithm.

For the L2 thresholding algorithm, quality is measured in terms of the number of peers correctly computing an alert *i.e.* the percentage of peers for whom $\left\|\overline{\mathcal{K}_i}\right\| < \epsilon$ when $\left\|\overline{\mathcal{G}}\right\| < \epsilon$, and the percentage of peers for whom $\left\|\overline{\mathcal{K}_i}\right\| \geq \epsilon$ when $\left\|\overline{\mathcal{G}}\right\| \geq \epsilon$. We measure the maximal, average and minimal quality over all the peers (averaged over a number of different experiments). Quality is reported in three different scenarios: overall quality, averaged over the entire experiment; and quality on stationary data, measured separately for periods in which the mean of the data is inside the ϵ-circle $\left(\left\|\overline{\mathcal{G}}\right\| < \epsilon\right)$ and for periods in which the means of the data is outside the circle $\left(\left\|\overline{\mathcal{G}}\right\| \geq \epsilon\right)$.

For the mean monitoring algorithm, quality is the average distance between $\overline{\mathcal{G}}$ and the computed mean vector $\overline{\mu}$. We plot, separately, the overall quality (during the entire experiment) and the quality after the broadcast phase ended.

Lastly, for the k-means algorithm, quality is defined as the distance between the solution of our algorithm and that computed by a centralized algorithm, given all the data of all of the peers.

We have measured the cost of the algorithm according to the frequency in which messages are sent by each peer. Because of the leaky bucket mechanism which is part of the algorithm, the rate of messages per average peer is bounded by two for every L time units (one to each neighbor, for an average of two neighbors per peer). The trivial algorithm that floods every change in the data would send messages at this rate. The communication cost of our algorithms is thus defined in terms of normalized messages - the portion of this maximal rate which the algorithm uses. Thus, 0.1 normalized messages means that nine times out of ten the algorithm manages to avoid sending a message. We report both overall cost, which includes the stationary and transitional phases of the experiment (and thus is necessarily higher), and the monitoring cost, which only refers to stationary periods. The monitoring cost is the cost paid by the algorithm even if the data remains stationary; hence, it measures the "wasted effort" of the algorithm. We also separate, where appropriate, messages pertaining to the computation of the L2 thresholding algorithm from those used for convergecast and broadcast of statistics.

There are many factors which may influence the performance of the algorithms. First, are those pertaining to the data: the number of dimensions d, the covariance σ, and the distance between the means of the Gaussians of the different epochs (the algorithm is oblivious to the actual values of the means), and the length of the epochs T. Second, there are factors pertaining to the system: the topology, the number of peers, and the size of the local data. Last, there are control arguments of the algorithm: most importantly ϵ – the desired alert threshold, and then also L – the maximal frequency of messages. In all the experiments that we report in this section, one parameter of the system was changed and the others were kept at their default values. The default values were : number of peers = 1000, $|X_{i,i}| = 800$, $\epsilon = 2$, $d = 5$, $L = 500$ (where the average edge delay is about 1100 time units), and the Frobenius norm of the covariance of the data $\|\sigma\|_F$ at 5.0. We selected the distance between the means so that the rates of false negatives and false positives are about equal. More specifically, the means for one of the epochs was +2 along each dimension and for the other it was -2 along each dimension. For each selection of the parameters, we ran the experiment for a long period of simulated time, allowing 10 epochs to occur.

A typical experiment is described in Figure 2(b) and 2(c). In the experiment, after every 2×10^5 simulator ticks, the data distribution is changed, thereby creating an epoch change. To start with, every peer is given the same mean as the mean of the Gaussian. Thus a very high percentage ($\sim$ 100 %) of the peers states that $\left\|\overline{\mathcal{G}}\right\| < \epsilon$. After the aforesaid number (2×10^5) of simulator ticks, we change the Gaussian without changing

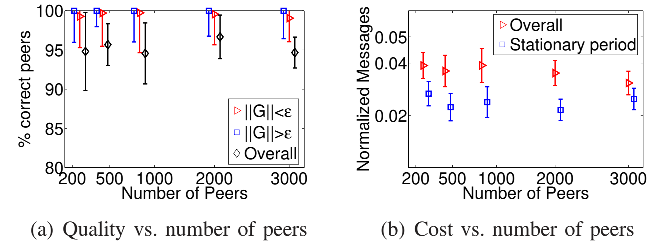
Algorithm 4 k-Means Monitoring

Input of peer p_i**:** ϵ, L, $X_{i,i}$, the set of immediate neighbors N_i, an initial guess for the centroids C_0, a mitigation constant τ, the sample size b.
Output of peer p_i**:** k centroids such that the average of the points assigned to every centroid is within ϵ of that centroid.
Data structure of peer p_i**:** A partitioning of $X_{i,i}$ into k sets $X^1_{i,i} \ldots X^k_{i,i}$, a set of centroids $C = \{\overline{c_1}, \ldots, \overline{c_k}\}$, for each centroid $j = 1, \ldots, k$, a flag $alert_j$, a times tamp $last_change_j$, a buffer B_j and a counter b_j, a flag $root$ and a flag $phase$.
Initialization:
Set $C \leftarrow C_0$. Let $X^j_{i,i} = \left\{ \overline{x} \in X_{i,i} : \overline{c_j} = \arg\min_{c \in C} \|\overline{x} - \overline{c}\| \right\}$. Initialize k instances of the L2 thresholding algorithm, such that the jth instance has input ϵ, α, L, $\left\{ \overline{x} - \overline{c_j} : \overline{x} \in X^j_{i,i} \right\}$, N_i. For all $p_j \in N_i$, set $b_j \leftarrow 0$, for all $j = 1, \ldots, k$ set $alert_j \leftarrow false$, $last_change_j \leftarrow -\infty$, and $phase \leftarrow convergecast$
On addition of a new vector $\overline{x}$ **to** $X_{i,i}$**:**
Find the c_j closest to $\overline{x}$ and add $\overline{x} - \overline{c_j}$ to the jth L2 thresholding instance.
On removal of a vector $\overline{x}$ **from** $X_{i,i}$**:**
Find the c_j closest to $\overline{x}$ and remove $\overline{x} - \overline{c_j}$ from the jth L2 thresholding instance.
On change in $\mathcal{F}\left(\overline{\mathcal{K}_i}\right)$ **of the** j^th **instance of the L2 thresholding algorithm:**
If $\left\|\overline{\mathcal{K}_i}\right\| \geq \epsilon$ and $alert_j = false$ then set $last_change_j \leftarrow time()$, $alert_j \leftarrow true$, and set a timer to τ time units
If $\left\|\overline{\mathcal{K}_i}\right\| < \epsilon$ then set $alert_j \leftarrow false$
On receiving B, b **from** $p_j \in N_i$**:**
Set $B_j \leftarrow B$, $b_j \leftarrow b$ and call Convergecast
On timer expiry or call to Convergecast:
If for all $\ell \in [1, \ldots, k]$ $alert_\ell = false$ then return
Let $t \leftarrow Min_{\ell=1\ldots k} \{last_message_\ell : alert_\ell = true\}$
Let A be a set of b samples returned by **Sample**
If $time() < t + \tau$ then set a timer to $t + \tau - time()$ and return
If for all $p_k \in N_i$ except for one $b_k \neq 0$
– Set $root \leftarrow false$, $phase \leftarrow Broadcast$
– Send $A, |X_{i,i}| + \sum_{m=1\ldots} b_m$ to p_ℓ and return
If for all $p_k \in N_i$ $b_k \neq 0$
– Let C' be the centroids resulting from computing the k-means clustering of A
– Set $root \leftarrow true$
– Send C' to self and return
On receiving C' **from** $p_j \in N_i$ **or from self:**
If $phase = convergecast$ and $i > j$ then return
Set $C \leftarrow C'$
For $j = 1 \ldots k$ set
$X^j_{i,i} = \left\{ \overline{x} \in X_{i,i} : \overline{c_j} = \arg\min_{c \in C} \|\overline{x} - \overline{c}\| \right\}$
For $j = 1 \ldots |N_i|$ set $b_j \leftarrow 0$
Send C to all $p_k \neq p_j \in N_i$
Set $phase \leftarrow Convergecast$
On call to Sample:
Return a random sample from $X_{i,i}$ with probability $1/\left(1 + \sum_{m=1\ldots|N_i|} b_m\right)$ or from a buffer B_j with probability $b_j/\left(|X_{i,i}| + \sum_{m=1\ldots|N_i|} b_m\right)$

the mean given to each peer. Thus, for the next epoch, we see that a very low percentage of the peers (~ 0 %) output that $\left\|\overline{\mathcal{G}}\right\| < \epsilon$. For the cost of the algorithm in Figure 2(c), we see that messages exchanged during the stationary phase is low. Many messages are, however, exchanged as soon as the epoch changes. This is expected since all the peers need to communicate in order to get convinced that the distribution has indeed changed. The number of messages decreases once the distribution becomes stable again.

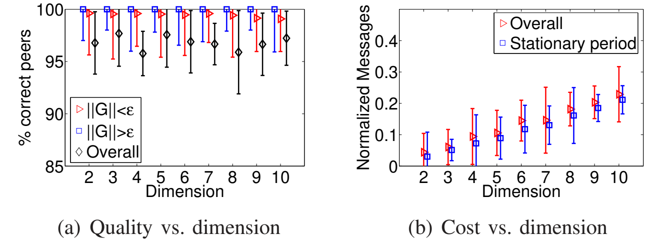
B. Experiments with Local L2 Thresholding Algorithm

The L2 thresholding algorithm is the simplest one we present here. In our experiments, we use the L2 thresholding to establish the scalability of the algorithms with respect to both the number of peers and the dimensionality of the data, and the dependency of the algorithm on the main parameters – the norm of the covariance σ, the size of the local data set, the tolerance ϵ, and the bucket size L.



(a) Quality vs. number of peers (b) Cost vs. number of peers

Fig. 3. Scalability of Local L2 algorithm with respect to the number of peers.



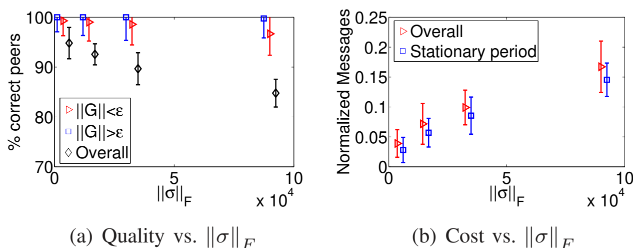
(a) Quality vs. dimension (b) Cost vs. dimension

Fig. 4. Scalability of Local L2 algorithm with respect to the dimension of the domain.

In Figures 3 and 4, we analyze the scalability of the local L2 algorithm. As Figure 3(a) and Figure 3(b) show, the average quality and cost of the algorithm converge to a constant as the number of peers increase. This typifies local algorithms – because the computation is local, the total number of peers do not affect performance. Hence, there could be no deterioration in quality or cost. Similarly, the number of messages per peer become a constant – typical to local algorithms. Figure 4(a) and Figure 4(b) show the scalability with respect to the dimension of the problem. As shown in the figures, quality does not deteriorate when the dimension of the problem is increased. Also note that the cost increases approximately linearly with the dimension. This independence of the quality can be explained if one thinks of what the algorithm does in terms of domain linearization. We hypothesis that when

the mean of the data is outside the circle, most peers tend to select the same half-space. If this is true then the problem is projected along the vector defining that half-space – i.e., becomes uni-dimensional. Inside the circle, the problem is again uni-dimensional: If thought about in terms of the polar coordinate system (rooted at the center of the circle), then the only dimension on which the algorithm depends is the radius. The dependency of the cost on the dimension stems from the linear dependence of the variance of the data on the number of Gaussians, the variance of whom is constant. This was proved in experiments not included here.

In Figures 5, 6, 7 and 8 we explore the dependency of the L2 algorithm on different parameters *viz.* Frobenius norm of the covariance of the data σ ($\|\sigma\|_F = \sum_{i=1...m} \sum_{j=1...n} |\sigma_{i,j}|^2$), the size of the local data buffer $|X_{i,i}|$, the alert threshold ϵ, and the size of the leaky bucket L. As noted earlier, in each experiment one parameter was varied and the rest were kept at their default values.

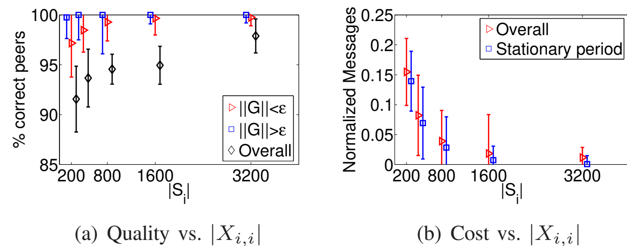


(a) Quality vs. $\|\sigma\|_F$ (b) Cost vs. $\|\sigma\|_F$

Fig. 5. Dependency of cost and quality of L2 thresholding on $\|\sigma\|_F$. Quality is defined by the percentage of peers correctly computing an alert (separated for epochs with $\|\overline{\mathcal{G}}\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported. Overall measurements include the transitional period too.

The first pair of figures, Figure 5(a) and Figure 5(b), outline the dependency of the quality and the cost on the covariance of the data ($\sigma = A\overline{E}$) where A is the covariance matrix and $\overline{E}$ is the variance of the gaussians. Matrix A is as defined in Section VI-A while $\overline{E}$ is the column vector representing the variance of the gaussians and takes the values 5, 10, 15 or 25. For epochs with $\|\overline{\mathcal{G}}\| < \epsilon$, the maximal, the average, and the minimal quality in every experiment decrease linearly with the variance (from around 99% on average to around 96%). Epochs with $\|\overline{\mathcal{G}}\| > \epsilon$, on the other hand, retained very high quality, regardless of the level of variance. The overall quality also decreases linearly from around 97% to 84%, apparently resulting from slower convergence on every epoch change. As for the cost of the algorithm, this increases as the square root of $\|\sigma\|_F$ (i.e., linear to the variance), both for the stationary and overall period. Nevertheless, even with the highest variance, the cost stayed far from the theoretical maximum of two messages per peer per leaky bucket period.

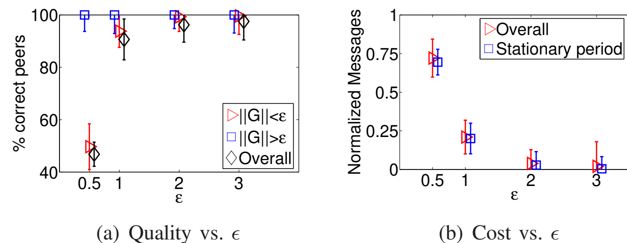
The second pair of figures, Figure 6(a) and Figure 6(b), shows that the variance can be controlled by increasing the local data. As $|X_{i,i}|$ increases, the quality increases, and cost decreases, proportional to $\sqrt{|X_{i,i}|}$. The cause of that is clearly the relation of the variance of an i.i.d. sample to the sample size which is inverse of the square root.



(a) Quality vs. $|X_{i,i}|$ (b) Cost vs. $|X_{i,i}|$

Fig. 6. Dependency of cost and quality of L2 thresholding on $|X_{i,i}|$. Quality is defined by the percentage of peers correctly computing an alert (separated for epochs with $\|\overline{\mathcal{G}}\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported. Overall measurements include the transitional period too.

The third pair of figures, Figure 7(a) and Figure 7(b), present the effect of changing ϵ on both the cost and quality of the algorithm. As can be seen, below a certain point, the number of false positives grows drastically. The number of false negatives, on the other hand, remains constant regardless of ϵ. When ϵ is about two, the distances of the two means of the data (for the two epochs) from the boundary of the circle are approximately the same and hence the rates of false positives and false negatives are approximately the same too. As ϵ decreases, it becomes increasingly difficult to judge if the mean of the data is inside the smaller circle and increasingly easier to judge that the mean is outside the circle. Thus, the number of false positives increase. The cost of the algorithm decreases linearly as ϵ grows from 0.5 to 2.0, and reaches nearly zero for $\epsilon = 3$. Note that even for a fairly low $\epsilon = 0.5$, the number of messages per peer per leaky bucket period is around 0.75, which is far less than the theoretical maximum of 2.

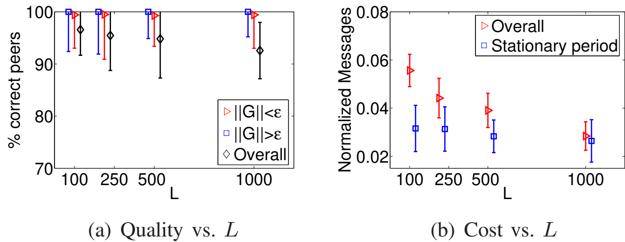


(a) Quality vs. ϵ (b) Cost vs. ϵ

Fig. 7. Dependency of cost and quality of L2 thresholding on ϵ. Quality is defined by the percentage of peers correctly computing an alert (separated for epochs with $\|\overline{\mathcal{G}}\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported. Overall measurements include the transitional period too.

Figure 8(a) and Figure 8(b) explore the dependency of the quality and the cost on the size of the leaky bucket L. Interestingly, the reduction in cost here is far faster than the reduction in quality, with the optimal point (assuming 1:1 relation between cost and quality) somewhere between 100 time units and 500 time units. It should be noted that the average delay BRITE assigned to an edge is around 1100 time units. This shows that even a very permissive leaky bucket mechanism is sufficient to greatly limit the number

of messages.



(a) Quality vs. L

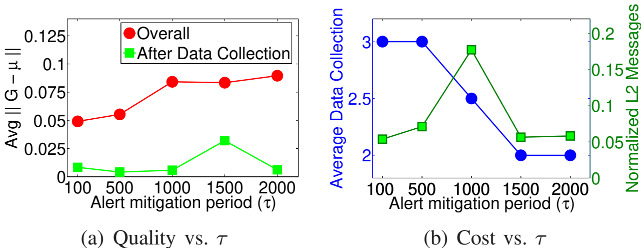
(b) Cost vs. L

Fig. 8. Dependency of cost and quality of L2 thresholding on L. Quality is defined by the percentage of peers correctly computing an alert (separated for epochs with $\|\overline{\mathcal{G}}\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported. Overall measurements include the transitional period too.

We conclude that the L2 thresholding provides a moderate rate of false positives even for noisy data and an excellent rate of false negatives regardless of the noise. It requires little communication overhead during stationary periods. Furthermore, the algorithm is highly scalable – both with respect to the number of peers and dimensionality – because performance is independent of the number of peers and dimension of the problem.

C. Experiments with Means-Monitoring

Having explored the effects of the different parameters of the L2 thresholding algorithm, we now shift our focus on the experiments with the mean monitoring algorithm. We have explored the three most important parameters that affect the behavior of the mean monitoring algorithm: τ – the alert mitigation period, T – the length of an epoch, and ϵ – the alert threshold.



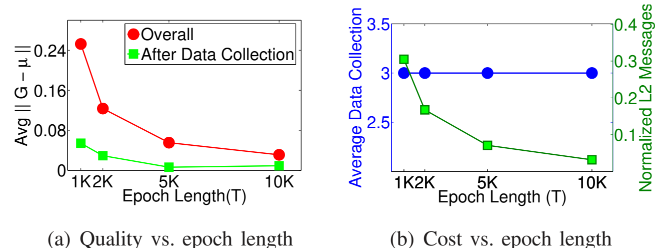
(a) Quality vs. τ

(b) Cost vs. τ

Fig. 9. Dependency of cost and quality of mean monitoring on the alert mitigation period τ.

Figure 9, 10 and 11 summarize the results of these experiments. As can be seen, the quality, measured by the distance of the actual means vector $\overline{\mathcal{G}}$ from the computed one $\overline{\mu}$ is excellent in all three graphs. Also shown are the cost graphs with separate plots for the L2 messages (on the right axis) and the number of convergecast rounds – each costs two messages per peer on average – (on the left axis) per epoch.

In Figure 9(a), the average distance between $\overline{\mathcal{G}}$ and $\overline{\mu}$ decreases as the alert mitigation period (τ) is decreased for the entire length of the experiment. This is as expected, since, with a smaller τ, the peers can rebuild the model more frequently, resulting in more accurate models. On the other hand, the quality after the data collection is extremely good and is independent of τ. With increasing τ, the number of convergecast rounds per epoch decreases (from three to two on average) as shown in Figure 9(b). In our analysis, this results from a decrease in the number of false alerts.



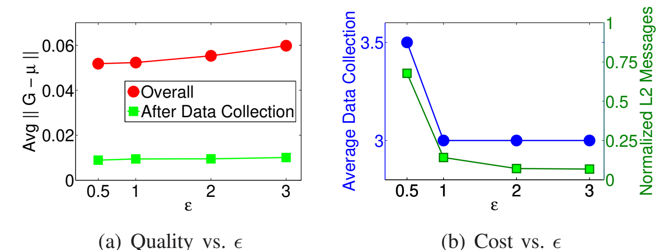
(a) Quality vs. epoch length

(b) Cost vs. epoch length

Fig. 10. Dependency of cost and quality of mean monitoring on the length of epoch T.

Figure 10(a) depicts the relation of the quality (both overall and stationary periods) to T. The average distance between the estimated mean vector and the actual one decreases as the epoch length T increases. The reason is the following: at each epoch, several convergecast rounds usually occur. The later the round is, the less polluted is the data by remnants of the previous epoch – and thus the more accurate is $\overline{\mu}$. Thus, when the epoch length increases, the proportion of these later $\overline{\mu}$'s, which are highly accurate, increases in the overall quality leading to a more accurate average. Figure 10(b) shows a similar trend for the cost incurred. One can see that the number of L2 messages decrease as T increases. Clearly, the more accurate $\overline{\mu}$ is, the less monitoring messages are sent. Therefore with increasing T, the quality increases and cost decreases in the later rounds and these effects are reflected in the figures.

Finally, the average distance between $\overline{\mathcal{G}}$ and $\overline{\mu}$ decreases as ϵ decreases. This is as expected, since with decreasing ϵ, the L2 algorithm ensures that these two quantities be brought closer to each other and thus the average distance between them decreases. The cost of the algorithm, however, shows the reverse trend. This result is intuitive – with increasing ϵ, the algorithm has a larger region in which to bound the global average and thus the problem becomes easier, and hence less costly, to solve.



(a) Quality vs. ϵ

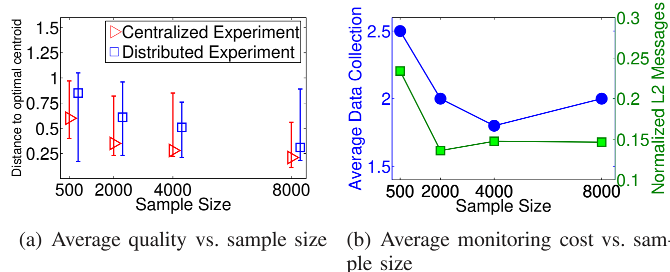
(b) Cost vs. ϵ

Fig. 11. Dependency of cost and quality of mean monitoring on the alert threshold ϵ.

On the whole, quality of the mean monitoring algorithm outcome behaves well with respect to all the three parameters influencing it. The monitoring cost *i.e.* L2 messages is also low. Furthermore, on an average, the number of convergecast

rounds per epoch is around three – which can easily be reduced further by using a longer τ as the default value.

D. Experiments with k-Means Monitoring



(a) Average quality vs. sample size (b) Average monitoring cost vs. sample size

Fig. 12. Dependency of quality and cost of k-means monitoring on the sample size

In this set of experiments our goal is to investigate the effect of the sample size on the k-means monitoring algorithm. To do that we compare the results of our algorithm to those of a centralized algorithm that processed the entire data. We compute the distance between each centroid computed by the peer-to-peer algorithm and the closest centroid computed by the centralized one. Since our algorithm is not only distributed but also sample-based, we include for comparison the results of centralized algorithm which takes a sample from the entire data as its input. The most outstanding result, seen in Figure 12(a), is that most of the error of the distributed algorithm is due to sampling and not due to decentralization. The error, both average, best case, and worst case, is very similar to that of the centralized sample-based algorithm. This is significant in two ways. First, the decentralized algorithm is obviously an alternative to centralization; especially considering the far lower communication cost. Secondly, the error of the decentralized algorithm can be easily controlled by increasing the sample size.

The costs of k-means monitoring have to be separated to those related to monitoring the current centroids and those related to the collection of the sample. Figure 12(b) presents the costs of monitoring a single centroid and the number of times data was collected per epoch. These could be multiplied by k to bound the total costs (note that messages relating to different centroids can be piggybacked on each other). The cost of monitoring decreases drastically with increasing sample size – resulting from the better accuracy provided by the larger sample. Also there is a decrease in the number of convergecast rounds as the sample size increases. The default value of the alert mitigation factor τ in this experimental setup was 500. For any sample size greater than 2000, the number of convergecast rounds is about two per epoch – in the first round, it seems, the data is so much polluted by data from the previous epoch that a new round is immediately triggered. As noted earlier, this can be further decreased using a larger value of τ.

VII. Related Work

Algorithms for large distributed systems have been developed over the last half decade. These can be roughly classified into three categories: convergecast based or centralized algorithms, gossip based algorithms, and local algorithms. Some best-effort heuristics [11], [12], [13] were suggested as well.

The first category, convergecast based algorithms, is perhaps the simplest. Algorithms such as [14] provide generic solutions – suitable for the computation of multiple functions. They are also extremely communication efficient: computing the average, for instance, only requires one message from each peer. Some of these algorithms can be extremely synchronized – every round of computation taking a lot of time. This becomes very problematic when the data is dynamic and computation has to be iterated frequently. Other, such as STAR [15] can dynamically tune accuracy and timeliness vs. communication overhead. The most thorough implementation of this approach is possibly the Astrolabe system [16] which implement a general purpose infrastructure for distributed system monitoring.

The second category, gossip based algorithms, relies on the properties of random walks on graphs to provide probabilistic estimates for various statistics of data stored in the graph. Gossip based computation was first introduced by Kempe *et al.* [17], and have, since then, been expanded to general graphs by Boyd *et al.* [18]. The first gossip based algorithms required that the algorithm be executed from scratch if the data changes in order to maintain those guarantees. This problem was later addressed by Jelasity *et al.* [19]. The main benefit of our algorithm with respect to gossiping is that it is data driven. Thus, it is far more efficient than gossiping when the changes are stationary.

Local algorithms were first discussed by Afek *et al.* [20], Linial [21], and Naor and Stockmeyer [22], in the context of graph theory. Kutten and Peleg introduced local algorithms in which the input is data which is stored at the graph vertices, rather than the graph itself [23]. The first application of local algorithms to peer-to-peer data mining is the Majority-Rule algorithm by Wolff and Schuster [1]. Since then, local algorithms were developed for other data mining tasks *e.g.*, decision tree induction [24], multivariate regression [6], outlier detection [3], L2 norm monitoring [4], approximated sum [25], and more. The algorithm for L2 thresholding, and an initial application of that algorithm for k-means monitoring were first presented in a previous publication by the authors of this paper [4].

VIII. Conclusions and Open Questions

In this paper we present a generic algorithm which can compute *any* ordinal function of the average data in large distributed system. We present a number of interesting applications for this generic algorithm. Besides direct contributions to the calculation of L2 norm, the mean, and k-means in peer-to-peer networks, we also suggest a new reactive approach in which data mining models are computed by an approximate or heuristic method and are then efficiently judged by an efficient local algorithm.

This work leaves several interesting open questions. The first is the question of describing the "hardness" of locally computing a certain function $\mathcal{F}$ – its "locallability". For

instance, it is simple to show that majority voting lends itself better for local computation than the parity function. However, there is lack of an orderly method by which the hardness of these and other functions can be discussed. The second interesting question is the question of robustness of a generic local algorithm for general topologies. Last, in view of our generic algorithm it would be interesting to revisit Naor's and Stockmeyer's question [22] regarding the limitations of local computation.

ACKNOWLEDGMENTS

This research is supported by the United States National Science Foundation CAREER award IIS-0093353 and NASA Grant NNX07AV70G.

REFERENCES

[1] R. Wolff and A. Schuster, "Association Rule Mining in Peer-to-Peer Systems," in *Proceedings of ICDM'03*, Melbourne, Florida, 2003, pp. 363–370.

[2] D. Krivitski, A. Schuster, and R. Wolff, "A Local Facility Location Algorithm for Sensor Networks," in *Proceedings of DCOSS'05*, Marina del Rey, California, 2005, pp. 368–375.

[3] J. Branch, B. Szymanski, R. Wolff, C. Gianella, and H. Kargupta, "In-Network Outlier Detection in Wireless Sensor Networks," in *Proceedings of ICDS'06*, Lisboa, Portugal, 2006, pp. 51–58.

[4] R. Wolff, K. Bhaduri, and H. Kargupta, "Local L2 Thresholding based Data Mining in Peer-to-Peer Systems," in *Proceedings of SDM'06*, Bethesda, Maryland, 2006, pp. 428–439.

[5] P. Luo, H. Xionga, K. Lu, and Z. Shi, "Distributed classification in peer-to-peer networks," in *Proceedings of SIGKDD'07*, San Jose, California, 2007, pp. 968–976.

[6] K. Bhaduri and H. Kargupta, "An Efficient Local Algorithm for Distributed Multivariate Regression in Peer-to-Peer Networks," in *Proceedings of SDM'08*, Atlanta, Georgia, 2008, pp. 153 – 164.

[7] N. Li, J. C. Hou, and L. Sha, "Design and Analysis of an MST-based Topology Control Algorithm," *IEEE Transactions on Wireless Communications*, vol. 4, no. 3, pp. 1195–1206, 2005.

[8] Y. Birk, L. Liss, A. Schuster, and R. Wolff, "A Local Algorithm for Ad Hoc Majority Voting Via Charge Fusion," in *Proceedings of DISC'04*, Amsterdam, Netherlands, 2004, pp. 275–289.

[9] K. Bhaduri, "Efficient Local Algorithms for Distributed Data Mining in Large Scale Peer to Peer Environments: A Deterministic Approach," Ph.D. dissertation, University of Maryland, Baltimore County, Baltimore, Maryland, USA, May 2008.

[10] K. Das, K. Bhaduri, K. Liu, and H. Kargupta, "Distributed Identification of Top-l Inner Product Elements and its Application in a Peer-to-Peer Network," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 20, no. 4, pp. 475–488, 2008.

[11] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, and S. Datta, "Clustering Distributed Data Streams in Peer-to-Peer Environments," *Information Science*, vol. 176, no. 14, pp. 1952–1985, 2006.

[12] W. Kowalczyk, M. Jelasity, and A. E. Eiben, "Towards Data Mining in Large and Fully Distributed Peer-to-Peer Overlay Networks," in *Proceedings of BNAIC'03*, Nijmegen, Netherlands, 2003, pp. 203–210.

[13] S. Datta, C. Giannella, and H. Kargupta, "K-Means Clustering over Large, Dynamic Networks," in *Proceedings of SDM'06*, Maryland, 2006, pp. 153–164.

[14] M. Rabbat and R. Nowak, "Distributed Optimization in Sensor Networks," in *Proceedings of IPSN'04*, California, 2004, pp. 20–27.

[15] N. Jain, D. Kit, P. Mahajan, P. Yalagandula, M. Dahlin, and Y. Zhang, "STAR: Self-tuning aggregation for scalable monitoring," in *Proceedings of VLDB'07*, Sept. 2007, pp. 962–973.

[16] R. van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 2, pp. 164–206, 2003.

[17] D. Kempe, A. Dobra, and J. Gehrke, "Computing Aggregate Information using Gossip," in *Proceedings of FOCS'03*, Cambridge, Massachusetts, 2003, pp. 482–491.

[18] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, " Gossip Algorithms: Design, Analysis and Applications," in *Proceedings of INFOCOM'05*, Miami, Florida, 2005, pp. 1653–1664.

[19] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based Aggregation in Large Dynamic Networks," *ACM Transactions on Computer Systems*, vol. 23, no. 3, pp. 219 – 252, 2005.

[20] Y. Afek, S. Kutten, and M. Yung, "Local Detection for Global Self Stabilization," *Theoretical Computer Science*, vol. 186, no. 1-2, pp. 199–230, 1997.

[21] N. Linial, "Locality in Distributed Graph Algorithms," *SIAM Journal of Computing*, vol. 21, no. 1, pp. 193–2010, 1992.

[22] M. Naor and L. Stockmeyer, "What can be Computed Locally?" in *Proceedings of STOC'93*, 1993, pp. 184–193.

[23] S. Kutten and D. Peleg, "Fault-Local Distributed Mending," in *Proceedings of PODC'95*, Ottawa, Canada, 1995, pp. 20–27.

[24] K. Bhaduri, R. Wolff, C. Giannella, and H. Kargupta, "Distributed Decision Tree Induction in Peer-to-Peer Systems," *Statistical Analysis and Data Mining Journal*, vol. 1, no. 2, pp. 85–103, 2008.

[25] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani, "The Price of Validity in Dynamic Networks," in *Proceedings of SIGMOD'04*, Paris, France, 2004, pp. 515–526.

Ran Wolff is faculty of the Management Information Systems department at University of Haifa, Israel. A graduate of the Technion – Israel, he previously held a post doctoral position at the University of Maryland in Baltimore County. His main fields of expertise are data mining in large-scale distributed environments: peer-to-peer networks, grid systems, and wireless sensor networks, and privacy preserving data mining. Ran regularly serves as PC in ICDM, SDM and SIGKDD, and as a reviewer for the DMKD and TKDE journals, among other. More information about him can be found at http://mis.haifa.ac.il/˜rwolff.

Kanishka Bhaduri received his B.E. in Computer Science and Engineering from Jadavpur University, India in 2003 and PhD degree in Computer Science from University of Maryland Baltimore County in 2008. Currently he is a research scientist with Mission Critical Technologies Inc at NASA Ames Research Center. His research interests include distributed and P2P data mining, data stream mining, and statistical data analysis. Kanishka serves as a reviewer for many conferences and journals such as ICDM, SDM, PKDD, SIGKDD, TKDE, TMC and more. More information about him can be found at http://www.csee.umbc.edu/˜kanishk1.

Hillol Kargupta is an Associate Professor at the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County. He received his Ph.D. in Computer Science from University of Illinois at Urbana-Champaign in 1996. He is also a co-founder of AGNIK LLC, a ubiquitous data intelligence company. His research interests include distributed data mining, data mining in ubiquitous environment, and privacy-preserving data mining. Dr. Kargupta won a US National Science Foundation CAREER award in 2001 for his research on ubiquitous and distributed data mining. He has published more than 90 peer-reviewed articles in journals, conferences, and books. He is an associate editor of the IEEE Transactions on Knowledge and Data Engineering, the IEEE Transactions on Systems, Man, and Cybernetics, Part B, and the Statistical Analysis and Data Mining Journal. He regularly serves on the organizing and program committees of many data mining conferences. More information about him can be found at http://www.csee.umbc.edu/˜hillol.