Local L2-Thresholding Based Data Mining in Peer-to-Peer Systems

Ran Wolff* Kanishka Bhaduri† Hillol Kargupta‡

Abstract

In a large network of computers, wireless sensors, or mobile devices, each of the components (hence, peers) has some data about the global status of the system. Many of the functions of the system, such as routing decisions, search strategies, data cleansing, and the assignment of mutual trust, depend on the global status. Therefore, it is essential that the system be able to detect, and react to, changes in its global status.

Computing global predicates in such systems is usually very costly. Mainly because of their scale, and in some cases (e.g., sensor networks) also because of the high cost of communication. The cost further increases when the data changes rapidly (due to state changes, node failure, etc.) and computation has to follow these changes. In this paper we describe a two step approach for dealing with these costs. First, we describe a highly efficient *local* algorithm which detect when the L2 norm of the average data surpasses a threshold. Then, we use this algorithm as a feedback loop for the monitoring of complex predicates on the data – such as the data's k-means clustering. The efficiency of the L2 algorithm guarantees that so long as the clustering results represent the data (i.e., the data is stationary) few resources are required. When the data undergoes an epoch change – a change in the underlying distribution – and the model no longer represents it, the feedback loop indicates this and the model is rebuilt. Furthermore, the existence of a feedback loop allows using approximate and "best-effort" methods for constructing the model; if an ill-fit model is built the feedback loop would indicate so, and the model would be rebuilt.

1 Introduction

Sensor networks, peer-to-peer systems, and other large distributed systems, produce, store and process huge amounts of status data as a main part of their daily operation. The purpose of collecting that status data is often to build a model of the system, and then either provide it to an operator or automatically act upon the model. Examples for such use of global models include facility location in sensor networks [12], trust assignment in peer-to-peer file sharing [9], peer-to-peer data mining [18, 11, 6, 1].

Since data in those systems (specifically status data) is usually time varying it is important to keep the model up-to-date. This can be done by periodically recomputing the model, by using incremental algorithms, or – as we suggest here – by monitoring the status and recomputing the model only when it no longer represents the data. The latter approach can be far more efficient than the first one because it is reactive. If the data is usually stationary and if the cost of monitoring is far lower than the cost of recomputing the model then it pays to only invest resources in recomputation when a monitoring mechanism indicates of a change. The alternative, recomputing the model periodically, has to trade between risking inaccuracy whenever the distribution of the data changes and consuming many resources while the data remained stationary. The monitoring approach is thus a valid alternative to incremental algorithms which are in many cases absent or inefficient.

A variety of metrics can be used in order to determine the suitability of a model to data. These would include statistical tests (e.g., chi-square χ^2), information theoretic methods (e.g., the Kullback-Leibler divergence $D_{k||\ell}$ [13]), and other methods. In this work we focus on the L2 norm of the data. The L2 norm is no more than the square root of the sum of distances between the data and the model. Examples for the use of the L2 norm are ample in the data mining field (e.g., in principle component analysis, in regression models, and in clustering). The χ^2 norm can also be presented as the L2 norm of the data, after it has been normalized according to the mean. This is significant because χ^2 has been used for change detection – a main challenge of data mining in time varying environments.

Two approaches which were previously suggested for computation over peer-to-peer networks are gossiping and computation with bounded error. With gossiping [10, 7, 3] it has been shown that aggregates such as the sum, average, etc. can be computed by a process in which each peer repeatedly averages its data with other peers it chooses at random. Beside its large communication overhead, no gossiping algorithm presented to date for aggregate calculations with dynamically changing data or partial failure of the system during execution. The second approach, proposed by Bawa at el. [2], is to compute sum and count queries subject to a model of the network behavior and guaranteeing bounded error. However, as Bawa at el. explain, calculating sums using their method can be very costly when the data dynamically changes. Furthermore, the associated error bounds are large (up to a factor of two). A similar approach which was taken by Bandyopadhyay at el. is to

*CSEE Dept., UMBC, ranw@cs.umbc.edu
†CSEE Dept., UMBC, kanishk1@cs.umbc.edu
‡CSEE Dept., UMBC and Agnik LLC, hillol@cs.umbc.edu

sample the network and compute probabilistic bounds on the error [1]. Still, this work does not address dynamically changing data.

A third approach which was proposed to computation in peer-to-peer is based on *local* algorithms. Permissively defined, local algorithms are ones whose resource consumption is sometimes independent of system size. That is, an algorithm for a given problem is local if there exists a constant c such that for any size of the system N there are instances of the problem such that the time, CPU, memory, and communication requirements per peer are smaller than c. Therefore, the most appealing property of local algorithms is their unlimited scalability. Local algorithms have been presented in the context of graph algorithms [15, 14], and recently for data mining in peer-to-peer networks [18, 12]. Local algorithms guarantee eventual correctness – when the computation terminates each peer computes the same result it would have computed given the entire data.

Contributions: In this paper we present a method for bounding the L2 norm of the average (or sum) of input vectors. The algorithm we present greatly generalize previous work on local majority votes which is first described in [18]. It describes a local stopping rule which can be applied to *any convex set* in any domain. We further introduce an entirely new approach for the use of local algorithm for the monitoring of many complex functions (including ones which are not convex). In this approach, the complex function is first approximated. Then, a local algorithm is employed to judge whether the quality of the approximation is satisfactory. If that quality is not good enough then the data is resampled and a new approximation is computed. The closed control loop provided by the local algorithm permits paying little attention to the quality of sampling – if that is unsatisfactory, resampling would occur.

We demonstrate this new approach by applying it to two different problems: The first problem is finding the mean vector of distributed data streams when the data is piece-wise stationary – i.e., data which transients abruptly between long periods in which it constantly changes in a stationary way (hence, epochs). The second one is the classic problem of k-means clustering [16], applied for the same kind of piece-wise stationary distributed data streams. Monitoring and updating of models was suggested earlier, both in the context of streams [8], and of incremental data mining [5, 17]. However, to the best of our knowledge never in distributed setting, let alone in peer-to-peer mining.

The following section presents notations, and some prerequisite lemmas. Section 3 describes the algorithm for local L2 Thresholding. Following, in Section 4 we describe the application of our method to monitoring of means, and in Section 5 the application for k-means clustering. We outline the experiments we performed in Section 6, and draw conclusions in Section 7.

2 Notations and Preliminaries

Let $P_1, \ldots, P_N$ be a set of peers connected to one another via an underlying communication tree such that the set of P_i's neighbors, N_i, is known to P_i. Additionally, P_i is given a stream of points from $\mathbb{R}^2$, a selection of which constitutes its *local data* $S_{i,t} = \{\vec{x_{i,1}}, \vec{x_{i,2}}, \ldots\}$. We use $\vec{S_{i,t}} = \frac{1}{|S_{i,t}|} \sum_{\vec{x} \in S_{i,t}} \vec{x}$ to denote the average vector of $S_{i,t}$. Additionally, peers communicate with one another by sending weighted vectors. We denote the last vector sent by peer P_i to P_j $\vec{X_{i,j}}$ and the weight assigned to this vector $\omega_{i,j}$. Assuming reliable messaging, once the message is delivered both P_i and P_j know both $\vec{X_{i,j}}$, $\vec{X_{j,i}}$, $\omega_{i,j}$, and $\omega_{j,i}$. We assume every peer P_i is notified when its data $S_{i,t}$ changes, when a message is received, and when the set of its immediate neighbors, N_i, changes.

Our algorithm makes specific use of three additional sets vectors and weights. The *knowledge* of P_i is denoted $\vec{X_i}, \omega_i$ where $\omega_i = |S_{i,t}| + \sum_{P_j \in N_i} \omega_{j,i}$ and $\vec{X_i} = \frac{|S_{i,t}|}{\omega_i} \vec{S_{i,t}} + \sum_{P_j \in N_i} \frac{\omega_{j,i}}{\omega_i} \vec{X_{j,i}}$. The *agreement* of P_i and a neighbor $P_j \in N_i$ is denoted $\vec{X_{i \cap j}}, \omega_{i \cap j}$ where $\omega_{i \cap j} = \omega_{i,j} + \omega_{j,i}$ and $\vec{X_{i \cap j}} = \frac{\omega_{i,j}}{\omega_{i \cap j}} \vec{X_{i,j}} + \frac{\omega_{j,i}}{\omega_{i \cap j}} \vec{X_{i,j}}$. The *kept knowledge* of P_i with respect to a neighbor $P_j \in N_i$ is $\vec{X_{i \setminus j}}, \omega_{i \setminus j}$ where $\omega_{i \setminus j} = \omega_i - \omega_{i \cap j}$ and $\vec{X_{i \setminus j}} = \frac{\omega_i}{\omega_{i \setminus j}} \vec{X_i} - \frac{\omega_{i \cap j}}{\omega_{i \setminus j}} \vec{X_{i \cap j}}$. Last, we use $\vec{X_N}$ to denote the true average of the data over all peers $\vec{X_N} = \sum_{i=1 \ldots N} \sum_{\vec{x} \in S_{i,t}} \vec{x} / \sum_{i=1 \ldots N} |S_{i,t}|$.

Throughout this paper $\vec{X} \cdot \vec{Y}$ represents the dot product of $\vec{X}$ and $\vec{Y}$, and $\left\| \vec{X} \right\|$ represents the L2 norm of $\vec{X}$. Let $\hat{u_1}, \ldots, \hat{u_d}$ be evenly spaced unit vectors (for the case of $\mathbb{R}^2$, unit vectors such that the angle between each two is $\frac{2\pi}{d}$). For some vector $\vec{x}$ and constant ϵ we denote $\hat{uf_i}$ the first $\hat{u}$ in $\hat{u_1}, \ldots, \hat{u_d}$ such that for at least one neighbor – P_j – it holds that $\hat{u} \cdot \vec{X_{i \cap j}} \geq \epsilon$. I.e., $\hat{uf_i} = \arg \min_{\hat{u} \in \{\hat{u_1} \ldots \hat{u_d}\}} \left\{ \exists P_j \in N_i : \hat{u} \cdot \vec{X_{i \cap j}} \geq \epsilon \right\}$. If for all $\hat{u} \in \{\hat{u_1} \ldots \hat{u_d}\}$ and all P_j we have that $\hat{u} \cdot \vec{X_{i \cap j}} < \epsilon$ then $\hat{uf_i}$ is set to *nil*.

2.1 Preliminaries

We now describe several conditions which apply to *termination states* of a distributed algorithm. In a termination state, no more messages

traverse the network, and hence the state of the entire network can be described in terms of just the variables each peer has. Local algorithms rely on conditions which allow extending predicates on the states of the different peers onto predicates on the global data X_N. Specifically, we will describe conditions on the different X_i, $X_{i\cap j}$, and $X_{i\setminus j}$, subject to whom it can be determined that $\left\|\vec{X_N}\right\| \leq \epsilon$ or that $\left\|\vec{X_N}\right\| > \epsilon$.

LEMMA 2.1. *Let $G(V,E)$ be a graph. For each $v_i \in V$ let $\vec{S_{i,t}}$ a vector and $|S_{i,t}|$ be a weight associated with it. For every $(v_i,v_j) \in E$, let $\vec{X_{i,j}}, \omega_{i,j}$ be a pair of an arbitrary vector and an arbitrary weight. Let $\vec{X_i}$ be the knowledge of P_i, $\vec{X_{i\cap j}}$ be the agreement of P_i and P_j, and $\vec{X_{i\setminus j}}$ be the kept knowledge of P_i with respect to P_j, all as described above. Let A be a convex region in $\mathbb{R}^d$. If for all $v_i \in V$ and every $(v_i,v_j) \in E$ it holds that $\vec{X_{i\cap j}} \in A$ and either $\vec{X_{i\setminus j}} \in A$ or $\omega_{i\setminus j} = 0$, then $\vec{X_N} \in A$.*

Proof. In Appendix A.

Specifically, since the d dimensional hyperball of radius ϵ is a convex shape in $\mathbb{R}^d$ we have that if for **every** peer P_i and **each** neighbor P_j of P_i it holds that both $\left\|\vec{X_{i\cap j}}\right\|$ and $\left\|\vec{X_{i\setminus j}}\right\|$ are below ϵ (i.e., in the d dimensional hyperball of radius ϵ) then the norm of the average of all vectors $\left\|\vec{X_N}\right\|$ is also below ϵ.

In [18], a local majority vote algorithm is described, where the data of each peer is a point in $\mathbb{R}$ and the decision needed to be made whether the average of the data is greater or smaller than a threshold value λ. The main Lemma described there, transferred to the terminology of this paper, is that if for all P_i and every neighbor P_j of P_i $\vec{X_i} \geq \vec{X_{i\cap j}} \geq \lambda$ then $\vec{X_N} \geq \lambda$ and $\vec{X_i} \leq \vec{X_{i\cap j}} < \lambda$ then $\vec{X_N} < \lambda$ ($\vec{X_i}$ and $\vec{X_{i\cap j}}$ are comparable with λ because in [18] they are vectors in $\mathbb{R}^1$). A closer look this lemma permits us to rewrite it in the following generalized form:

LEMMA 2.2. *Let $\vec{\lambda}$ be an arbitrary vector in $\mathbb{R}^d$, and let $\lambda^+, \lambda^- \subset \mathbb{R}^d$ be the half-spaces such that $\lambda^+ = \left\{\vec{x} \in \mathbb{R}^d : \vec{\lambda}\cdot\vec{x} \geq \vec{\lambda}\cdot\vec{\lambda}\right\}$ and $\lambda^- = \left\{\vec{x} \in \mathbb{R}^d : \vec{\lambda}\cdot\vec{x} < \vec{\lambda}\cdot\vec{\lambda}\right\}$. If for every P_i and each neighbor P_j of P_i it holds that $\vec{X_{i\cap j}} \in \lambda^+$ and either $\vec{X_{i\setminus j}} \in \lambda^+$ or $\vec{X_i} = \vec{X_{i\cap j}}$ then $\vec{X_N} \in \lambda^+$, and if it holds for all P_i and each neighbor P_j of P_i that $\vec{X_{i\cap j}} \in \lambda^-$ and either $\vec{X_{i\setminus j}} \in \lambda^-$ or $\vec{X_i} = \vec{X_{i\cap j}}$ then $\vec{X_N} \in \lambda^-$.*

Proof. In the special case of half-spaces, both λ^+ and λ^- are convex. If $\vec{X_i} = \vec{X_{i\cap j}}$ then either $\vec{X_{i\setminus j}} = \vec{X_{i\cap j}}$ or $\omega_{i\setminus j} = 0$. In either case, correctness follows from Lemma 2.1.

COROLLARY 2.1. *Given d unit vectors, $\hat{u_1}\ldots\hat{u_d}$ if for a specific one of them $\hat{u}$ every peer P_i and each of its neighbors P_j have $\hat{u}\cdot\vec{X_{i\cap j}} \geq \epsilon$ and either $\hat{u}\cdot\vec{X_{i\setminus j}} \geq \epsilon$ or $\hat{u}\cdot\vec{X_i} = \hat{u}\cdot\vec{X_{i\cap j}}$ then $\left\|\vec{X_N}\right\| \geq \epsilon$.*

Proof. Taking $\vec{\lambda} = \epsilon\hat{u}$, it follows from Lemma 2.2 that $\hat{u}\cdot\vec{X_N} \geq \epsilon$. Since $\hat{X_N}\cdot\vec{X_N} \geq \hat{u}\cdot\vec{X_N}$ for all $\hat{u}$ it follows that $\left\|\vec{X_N}\right\| = \hat{X_N}\cdot\vec{X_N} \geq \epsilon$.

For evenly spaced unit vectors, the larger d is the smaller the chances are that although $\left\|\vec{X_N}\right\| \geq \epsilon$ all $\hat{u} \in \{\hat{u_1}\ldots\hat{u_d}\}$ have $\hat{u}\cdot\vec{X_N} < \epsilon$. However, increasing the number of unit vectors comes at a computational cost. lemma 2.3 shows that even if d is large, it is enough for each peer to focus on a single unit vector each time, the first one that has for some neighbor $\hat{u}\cdot\vec{X_{i\cap j}} \geq \epsilon - \hat{uf_i}$.

LEMMA 2.3. *Let $\hat{u_1},\ldots,\hat{u_d}$ be a set of unit vectors agreed upon among all peers and let $\hat{uf_i}$ be the first such vector for which P_i has for one of its neighbors P_j that $\hat{u}\cdot\vec{X_{i\cap j}} \geq \epsilon$. If for every peer P_i the respective $\hat{uf_i}$ provides that for each of its neighbors P_j the respective $\hat{uf_i}$ provides that $\hat{uf_i}\cdot\vec{X_{i\setminus j}}, uf_i\cdot\vec{X_{i\cap j}} \geq \epsilon$ then all of the peers have the same $\hat{uf_i}$.*

Proof. In Appendix A.

It follows immediately from Lemma 2.3 and Lemma 2.2 that in this case $\hat{uf_i}\cdot\vec{X_N} \geq \epsilon$.

3 Local L2 Norm Thresholding

Relying on the lemmas described in the previous section, we now describe an algorithm (Alg. 3.1) which decides whether the L2 norm of the average input vector $\vec{X_N}$ is smaller than a given threshold value ϵ. The idea of the algorithm can be demonstrated with Figure 1. Each peer P_i checks if its $\vec{X_i}$ is inside a circle of radius ϵ , outside the polygon defined by d evenly spaced vectors of length ϵ, or between the circle and the polygon. If $\vec{X_i}$ is inside a circle, the peer will bring itself to a state which satisfies Lemma 2.1. If $\vec{X_i}$ is outside the polygon, the peer will bring itself to a state that satisfies Lemma 2.2. If $\vec{X_i}$ is between the circle and the polygon then the peer cannot bring its state to one that satisfies any of those lemma, so it has to propagate any data it has to its neighbors. Together, these three cases guarantee that eventually either all peers satisfy Lemma 2.1, they all satisfy Lemma 2.2, or – in the worst case – they all have all of the data and thus compute $\vec{X_N}$ precisely. In any case, eventual correctness is guaranteed.

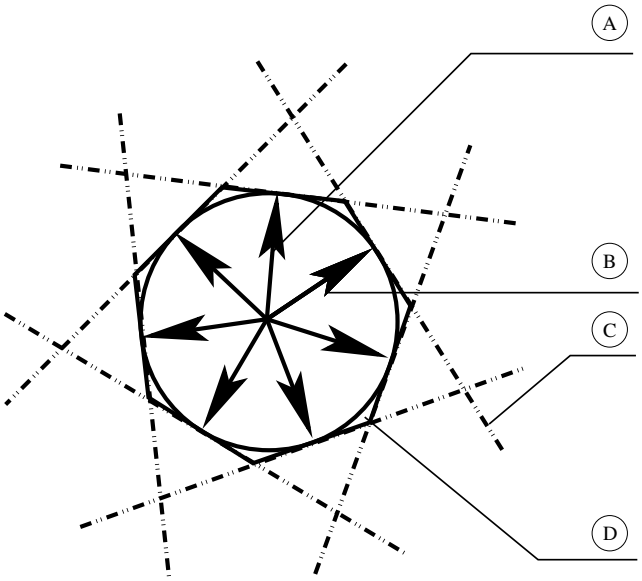


Figure 1: (A) the area inside an ϵ circle. (B) Seven evenly spaced vectors - $\vec{u_1} \dots \vec{u_7}$. (C) The borders of the seven halfspaces $\hat{u_i} \cdot \vec{x} \geq \epsilon$ define a polygon in which the circle is circumscribed. (D) The area between the circle and the union of half-spaces.

Given that $\vec{X_i}$ is in the circle, there would always be a way for P_i to move into a state that satisfies Lemma 2.1. All P_i has to do is to find any $P_j \in N_i$ for whom either $\vec{X_{i\cap j}}$ or $\vec{X_{i\setminus j}}$ is outside the circle. For any such P_j it can then send a message $\vec{X_{i,j}}, \omega_{i,j}$ which changes $\vec{X_{i\cap j}}$ and $\vec{X_{i\setminus j}}$. The values sent are computed such that $\omega_{i,j}$ is set to $\alpha(\omega_i - \omega_{j,i})$ and that after the message is sent $\vec{X_{i\cap j}} = \vec{X_{i\setminus j}} = \vec{X_i}$, where α is chosen between zero and one (see detailed exploration of the effect of different α values in the Section 6). Consequently, the conditions of Lemma 2.1 would now hold for P_i with respect to this neighbor. Of course, the message alters the vectors computed by P_j – $\vec{X_j}$, $\vec{X_{j\cap i}}$, and $\vec{X_{j\setminus i}}$– and thus, P_j may now be forced to send messages to either P_i or to other neighbors P_j may have.

Similarly, if P_i finds $\vec{X_i}$ to be outside the polygon, it acts to preserve the conditions of Lemma 2.2. It computes $\hat{uf_i}$ – the first unit vector $\hat{u} \in \{\hat{u_1}, \dots, \hat{u_d}\}$ such that for some neighbor $P_k \in N_i$ the dot product $\hat{u} \cdot \vec{X_{i\cap k}}$ is greater than or equal to ϵ. Then, if for any neighbor $P_j \in N_i$ $\hat{uf_i} \cdot \vec{X_{i\setminus j}} < \epsilon$ it sends a message to P_j which would, similarly to the previous case, set $\vec{X_{i\cap j}}$ and $\vec{X_{i\setminus j}}$ to $\vec{X_i}$ and $\omega_{i,j}$ to $\alpha(\omega_i - \omega_{j,i})$. Consequently, the conditions of Lemma 2.2 would thus hold for $\hat{uf_i}$and P_j. Again, that message would change P_j's data and may or may not trigger P_j to send additional messages.

Finally, there is a chance that $\vec{X_i}$ would neither be in the circle nor outside the polygon. In this case, the current state of P_i cannot a termination state according to any of the two Lemmas. Thus, the only option available to the algorithm is to flood the data. In this case, α is set to one, and a message is sent whenever $\vec{X_i} \neq \vec{X_{i\cap j}}$ or $\omega_i \neq \omega_{i\cap j}$. As a result, $\omega_{i\setminus j}$ is kept at zero, and $\vec{X_{i\setminus j}}$ is ill-defined (because of a devision by zero) – but is not used.

It is left to specify what a peer does on the events of receiving a message, a change in the local data, or a change in N_i due to neighbors leaving or joining in. Since $\vec{X_i}$ is defined as the weighted average of the vectors of P_i and ω_i as the sum of their weights, each such event affects $\vec{X_i}$ and ω_i: a received message changes $\vec{X_{j,i}}, \omega_{i,j}$ for the neighbor P_j from whom it was received; a change of the local data changes $\vec{S_{i,t}}, |S_{i,t}|$; and a change in N_i introduces a new component to the sum, or removes an existing one. The peer thus does nothing more than reevaluating its new state according to the described above, and issuing messages if they are required.

Last, since this algorithm is intended for asynchronous systems we include in it a *leaky bucket* mechanism. We restrict the frequency in which messages are sent to once every L time units. When a message needs to be sent, a peer first checks how long is it since the last message was sent. If less then L time units have passed, it waits the remaining time and then validate that the message still needs to be sent. Without a leaky bucket mechanism, and with messages arriving at totally random times, the number of messages in an event-based algorithm explodes.

ALGORITHM 3.1. Local L2 Thresholding

Input of peer P_i: ϵ, α, L, $S_{i,t}$, the set of immediate neighbors N_i
Output of peer P_i: $\left\|\vec{X_i}\right\|$
Data structure for P_i: For each $P_j \in N_i$ $\vec{X_{i,j}}$, $|X_{i,j}|$, $\vec{X_{j,i}}$, $|X_{i,j}|$, *last_message*
Initialization:
Compute $\hat{u_1}, \dots, \hat{u_d}$, *last_message* $\leftarrow -\infty$
On receiving a message $\vec{X}, |X|$ from P_j:
Set $\vec{X_{j,i}} \leftarrow \vec{X}$, $|X_{j,i}| \leftarrow |X|$
On change in $S_{i,t}$, N_i, $\vec{X_i}$ or $|X_i|$:
1. Let $\hat{uf_i}$ be the first of $\hat{u_1} \dots \hat{u_d}$ such that for some $\vec{X_{i\cap j}}$ it holds that $\hat{u_i} \cdot \vec{X_{i\cap j}} \geq \epsilon$, or *nil* if no such $\vec{X_{i\cap j}}$ exists
2. For each $P_j \in N_i$
– Case 1: $\left\|\vec{X_i}\right\| < \epsilon$
– – If $\left\|\vec{X_{i\setminus j}}\right\| \geq \epsilon$ or $\left\|\vec{X_{i\cap j}}\right\| \geq \epsilon$ SendMessage(P_j, α)

– Case 2: $\left\|\vec{X_i}\right\| \geq \epsilon$ and $\hat{uf_i}$ is not *nil*
– – If $\hat{uf_i} \cdot \vec{X_{i\cap j}} < \epsilon$ or $\hat{uf_i} \cdot \vec{X_{i\setminus j}} < \epsilon$ SendMessage(P_i, α)
– Case 3: $\left\|\vec{X_i}\right\| \geq \epsilon$ and $\hat{uf_i}$ is nil
– – If $\vec{X_i} \neq \vec{X_{i\cap j}}$ or $\omega_i \neq \omega_{i\cap j}$ SendMessage(P_j, 1)
SendMessage(P_j, β):
If $time\,() - last\_message < L$ then wait for $time\,() - last\_message$ time units and then call OnChange()
Set $\omega_{i,j} = \beta\,(\omega_i - \omega_{j,i})$
Set $\vec{X_{i,j}} \leftarrow \left(\omega_{i\cap j}\vec{X_i} - \omega_{j,i}\vec{X_{j,i}}\right)/\omega_{i,j}$
Set $last\_message \leftarrow time\,()$
Send $\vec{X_{i,j}}, \omega_{i,j}$ to P_j

4 Mean Monitoring

Having presented an efficient algorithm for L2 thresholding, we now turn to the different, but closely related computational task of monitoring the mean of the data. We describe an algorithm which computes, at every given time, a vector $\vec{\mu}$ which is guaranteed to track the mean vector $\vec{X_N}$. I.e., as $\vec{X_N}$ changes, $\vec{\mu}$ is guaranteed to quickly follow it until $\left\|\vec{\mu} - \vec{X_N}\right\| \leq \epsilon$. As Section 5 demonstrates, this has immediate applicability for clustering over distributed non-stationary data streams.

4.1 Algorithm The outcome of the L2 thresholding algorithm described in the previous section can be viewed as an alert flag at each peer, which is set in case $\left\|\vec{X_i}\right\| \geq \epsilon$. The guarantee of the L2 thresholding algorithm is that whenever the data stops changing, the algorithm converges until the flags of all peers are set if and only if $\left\|\vec{X_N}\right\| \geq \epsilon$. Assuming a guess $\vec{\mu}$ of the mean $\vec{X_N}$ is given to all peers, the same algorithm can be used to check whether $\left\|\vec{X_N} - \vec{\mu}\right\| \geq \epsilon$ by simply using the difference between every input vector $\vec{x_{i,j}}$ and $\vec{\mu}$ instead of the original $\vec{x_{i,j}}$.

If changes in the input are sparse enough for the algorithm to converge between them, then, theoretically, every time the algorithm converges to an alert, we could collect statistics of the data and approximate $\vec{X_N}$. Thus, a closed loop algorithm can be described which does so, and then updates $\vec{\mu}$ to the approximation of $\vec{X_N}$. By that, the peers would be *monitoring* the means of the global data, i.e., they would maintain an approximation of the means which is updated whenever the approximation becomes inaccurate.

This simplistic algorithm is lacking in several important aspects: First, a monitoring algorithm has better work even when changes are not sparse. Specifically, we would want it to work when stationary changes to the data are frequent and changes to the underlying distribution – *epoch changes* – are rare. Secondly, no algorithm can rely on the convergence of the L2 thresholding algorithm for anything but its eventual correctness. This is because the L2 thresholding algorithm provides the peers with no indication of termination.

Therefore, the Mean Monitoring algorithm (Alg. 4.1) slightly deviates from the simplistic one. Instead of waiting for the algorithm to converge, each peer which is alerted waits for a predefined measure of time, τ. If during this time the alert has remained, then it is considered a stable alert and is acted upon. Notably, the L2 thresholding algorithm does provide that a false alert would be removed eventually.

The rest of the algorithm is as follows: Each peer participates in a convergecast algorithm – it waits until it receives sufficient statistics (i.e., an average vector) from all but one of its neighbors. When it gets those statistics, and only if it observes a stable alert, it combines its own data with the statistics and forwards the result to the remaining neighbor. A peer that received statistics from all of its neighbors becomes a root. It combines the statistics it got from its neighbors and those of its own data to compute the new $\vec{\mu}$ and then sends it to its neighbors. Note there could be at most two roots, and that they collect the exact same statistics and compute the exact same $\vec{\mu}$. A peer who receives the new $\vec{\mu}$ from a neighbor forwards it to its other neighbors. It updates its data by subtracting the new $\vec{\mu}$ from its input and is now ready for an additional round of convergecast, if one is needed.

Another point to notice is that if the statistics collected in the convergecast are based on the same data which was used to compute the alert then they might be biased. Arguably, a peer with outlied data is more probable to alert and thus outlied data is more probable to be collected as statistics. To prevent that bias, every new data point is randomly put into an alert buffer or a statistics buffer. The points in the alert buffer are the ones that are used by the L2 thresholding algorithm. Those in the statistics buffer are the ones which are used by the convergecast algorithm.

ALGORITHM 4.1. Mean Monitoring

Input of peer P_i:
ϵ, α, L, $S_{i,t}$, the set of immediate neighbors N_i, an initial vector $\vec{\mu_0}$, an alert mitigation constant τ.
Output of peer P_i:
An approximated means vector $\vec{\mu}$ such that $\left\|\vec{X_N} - \vec{\mu}\right\| < \epsilon$
Data structure of peer P_i:

Two sets of vectors $R_{i,t}$ and $T_{i,t}$, a flag *alert*, a timestamp *last_change*, for each $P_j \in N_i$, a vector $\vec{v}_j$ and a counter c_j

Initialization:

Set $\vec{\mu} \leftarrow \vec{\mu_0}$, $alert \leftarrow false$

Split $S_{i,t}$ evenly between $R_{i,t}$ and $T_{i,t}$

Initialize an L2 thresholding algorithm with the input ϵ, α, L,$\{\vec{x_{i,1}} - \vec{\mu}, \vec{x_{i,2}} - \vec{\mu}, \ldots, \vec{x_{i,B}} - \vec{\mu} : \vec{x_{i,j}} \in R_{i,t}\}$, N_i

Set $\vec{v_j}, c_j$ to $\vec{0}, 0$ for all $P_j \in N_i$

On addition of a new vector $\vec{x}$ to $S_{i,t}$:

Randomly add $\vec{x}$ to either $R_{i,t}$ or $T_{i,t}$ and retire the oldest point in the corresponding set.

If $\vec{x}$ was added to $R_{i,t}$, pass $\vec{x} - \vec{\mu}$ to the L2 thresholding algorithm.

On change in $\vec{X_i}$ of the L2 thresholding algorithm:

Case 1: $\left\|\vec{X_i}\right\| \geq \epsilon$

– If $alert = false$ then set $last\_change \leftarrow time()$, $alert \leftarrow true$

Case 2: $\left\|\vec{X_i}\right\| < \epsilon$

– Set $alert \leftarrow false$

On receiving $\vec{v}, c$ from $P_j \in N_i$:

Set $\vec{v_j} \leftarrow \vec{v}$, $c_j \leftarrow c$

If for all $P_k \in N_i$ except for one $c_k \neq 0$

– Wait while $alert = false$ or $time() - last\_change < \tau$

– If for all $P_k \in N_i$ except for one $(P_l) c_k \neq 0$

– – Let $s = |T_{i,t}| + \sum_{P_j \in N_i} c_j$,

$\vec{s} = \frac{|T_{i,t}|}{s} \vec{T_{i,t}} + \sum_{P_j \in N_i} \frac{c_j}{s} \vec{v_j}$

– – Send $s, \vec{s}$ to P_l

– If for all $P_k \in N_i$ $c_k \neq 0$

– – Let $s = |T_{i,t}| + \sum_{P_j \in N_i} c_j$,

$\vec{s} = \frac{|T_{i,t}|}{s} \vec{T_{i,t}} + \sum_{P_j \in N_i} \frac{c_j}{s} \vec{v_j}$

– – Set $\vec{\mu}$ to all $P_k \in N_i$

– – For all $\vec{x} \in R_{i,t}$, pass $\vec{x} - \mu$ to the L2 Thresholding algorithm

On receiving $\vec{\mu'}$ from $P_j \in N_i$:

Set $\vec{\mu} \leftarrow \vec{\mu'}$

Send $\vec{\mu}$ to all $P_k \neq P_j \in N_i$

For all $\vec{x} \in R_{i,t}$, add $\vec{x} - \mu$ to the $S_{i,t}$ of the L2 Thresholding algorithm

5 k-Means Clustering

The final algorithm we describe is a variant of the popular k-means clustering algorithm. To make k-means suitable for peer-to-peer networks and for dynamically changing data we make one slight change in the stopping rule of k-means: Instead of stopping when each centroid is exactly the mean of the points clustered to it, we define an admissible solution as one in which the distance between the centroid and the mean of the points clustered to it is less than a constant ϵ. The basic idea of the algorithm is to randomly sample the data from the network and cluster the sample rather than the entire data. Then, use L2 Thresholding to make certain that the distance between the centroids computed from the sample and the means of all of the data points clusters to these centroids is small. The local nature of the L2 Thresholding algorithm allows computing this efficiently. On the other hand, the feedback mechanism implemented through L2 Thresholding allows collection of very small samples. This is because if this sample fails to properly represent the data then L2 Thresholding will simply trigger another round of sample collection. Optionally, the size of the sample can also be increased the second time around.

5.1 Algorithm Just like centralized k-means clustering, the peer-to-peer k-means clustering algorithm (Alg. 5.1) begins with a guess of the location of the k centroids. We denote the set of locations $C = \{\vec{c_1}, \ldots, \vec{c_k}\}$, and maintain that it is equal for all peers. Following, every peer separates the points in its local data $S_{i,t}$ to clusters $S^1_{i,t}, \ldots S^k_{i,t}$ such that each point $\vec{x} \in S_{i,t}$ is allocated to the centroid nearest to it $S^j_{i,t} = \left\{\vec{x} \in S_{i,t} : \vec{c_j} = \arg\min_{\ell=1\ldots k} \|\vec{x} - \vec{c_\ell}\|\right\}$. Note that this allocation can be computed by every single peer and that given that the data is dynamic, points may be added and removed at every time, and the $S^j_{i,t}$ updated respectively.

The algorithm proceeds by concurrently executing an instance of L2 Thresholding per centroid. The data of every such instance is the respective $S^j_{i,t}$, modified by reducing the centroid $\vec{c_j}$ from each of the points. In case of a stable alert by any instance of the L2 Thresholding algorithm, the peer which is alerted participates in convergecast of a sample of the data points: It waits for samples from all but one of its neighbors. Then, it creates a new sample which blends its own data with the samples of its neighbors, proportionally to the number of peers represented in each sample received. It then passes the new sample to the remaining neighbor. A peer who receives samples from all of its neighbors (hence, a root), again blends in its own data and produces a uniform sample.

The root then uses this sample as the input for unmodified, centralized k-means: It iteratively first cluster the data points in the sample to their nearest centroids (starting with the current ones as an initial guess) and then moves the centroids to the means of the data points allocated to them – until *no further change* occurs. Finally, the root sends the computed centroids to its neighbors. A peer who receives a new set of centroids repartitions $S_{i,t}$, according to these centroids

and respectively updates the inputs of the instances of the L2 Thresholding algorithm to the new $S^j_{i,t}$, modified according to the new location of $\vec{c_j}$.

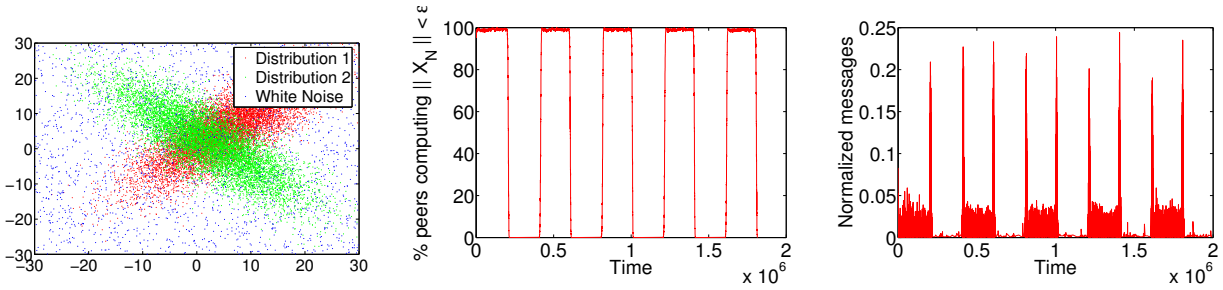
Note that unlike the Mean-Monitoring algorithm, no mechanism is employed here to avoid the bias in the sample. This is due to the way we define the problem – we wish the computed means to be a valid (within ϵ) representation of the entire data – and not just of a sample from it. This, however, only hinders performance, not correctness, as a biased sample which results in a less representative centroids would automatically trigger resampling.

ALGORITHM 5.1. k-Means Clustering in Peer-to-Peer

Input of peer P_i:
ϵ, α, L, $S_{i,t}$, 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 the centroid.
Data structure of peer P_i:
A partitioning of $S_{i,t}$ into k sets $S^1_{i,t} \dots S^k_{i,t}$, a set of centroids $C = \{\vec{c_1}, \dots, \vec{c_k}\}$, for each centroid $j = 1, \dots, k$, a flag $alert_j$, a times tamp $last\_change_j$, a buffer B_j and a counter b_j
Initialization:
Set $C \leftarrow C_0$, for $j = 1 \dots k$
$S^j_{i,t} = \left\{ \vec{x} \in S_{i,t} : \vec{c_j} = \arg\min_{\ell=1\dots k} \|\vec{x} - \vec{c_\ell}\| \right\}$
Initialize k instances of the L2 thresholding algorithm, such that the jth instance has input ϵ, α, L, $\left\{ \vec{x} - \vec{c_j} : \vec{x} \in S^j_{i,t} \right\}$, N_i
For all $P_j \in N_i$, set $b_j \leftarrow 0$, for all $j = 1, \dots, k$ set $alert_j \leftarrow false$, $last\_change_j \leftarrow -\infty$
On addition of a new vector $\vec{x}$ to $S_{i,t}$:
Find the c_j closest to $\vec{x}$ and add $\vec{x} - \vec{c_j}$ to the jth L2 Thresholding instance.
On removal of a vector $\vec{x}$ from $S_{i,t}$:
Find the c_j closest to $\vec{x}$ and remove $\vec{x} - \vec{c_j}$ from the jth L2 Thresholding instance.
On change in $\vec{X_i}$ of the jth instance of the L2 thresholding algorithm:
Case 1: $\left\|\vec{X_i}\right\| \geq \epsilon$
– If $alert_j = false$ then set $last\_change_j \leftarrow time()$, $alert_j \leftarrow true$
Case 2: $\left\|\vec{X_i}\right\| < \epsilon$
– Set $alert_j \leftarrow false$
On receiving B, b from $P_j \in N_i$:
Set $B_j \leftarrow B$, $b_j \leftarrow b$
If for all $P_k \in N_i$ except for one $b_k \neq 0$
– Wait while for some $\ell = 1, \dots, k$ either $alert_\ell = false$ or $time() - last\_change_\ell < \tau$
– If for all $P_k \in N_i$ except for one (P_ℓ) $b_k > 0$
– – Let A be a sample of size b from $S_{i,t}$ and B_1 through $B_{|N_i|}$ such that every point in B_j is sampled with repetitions at probability $b_j / (1 + \sum_{m=1\dots} b_m)$ and points in $S_{i,t}$ are sampled with repetition at probability $1/(1 + \sum_{m=1\dots} b_m)$
– – Send $A, 1 + \sum_{m=1\dots} b_m$ to P_ℓ
– If for all $P_k \in N_i$ $b_k \neq 0$
– – Let A be a sample of size b from $S_{i,t}$ and B_1 through $B_{|N_i|}$ such that every point in B_j is sampled with repetitions at probability $b_j / (1 + \sum_{m=1\dots} b_m)$ and points in $S_{i,t}$ are sampled with repetition at probability $1/(1 + \sum_{m=1\dots} b_m)$
– – Compute k-means clustering of A with the initial centroids set at C.
– – Set C to the resulting centroids. For $j = 1 \dots k$
$S^j_{i,t} = \left\{ \vec{x} \in S_{i,t} : \vec{c_j} = \arg\min_{\ell=1\dots k} \|\vec{x} - \vec{c_\ell}\| \right\}$
– – Remove all points from the data of the L2 Thresholding algorithm instances, and for $j = 1 \dots k$ pass $\left\{ \vec{x} - \vec{c_j} : \vec{x} \in S^j_{i,t} \right\}$ to the jth instance of the L2 Thresholding algorithm
– – Send C to all $P_k \in N_i$
On receiving C' from $P_j \in N_i$:
Set $C \leftarrow C'$
For $j = 1 \dots k$
$S^j_{i,t} = \left\{ \vec{x} \in S_{i,t} : \vec{c_j} = \arg\min_{\ell=1\dots k} \|\vec{x} - \vec{c_\ell}\| \right\}$
Remove all points from the data of the L2 Thresholding algorithm instances, and for $j = 1 \dots k$ pass $\left\{ \vec{x} - \vec{c_j} : \vec{x} \in S^j_{i,t} \right\}$ to the jth instance of the L2 Thresholding algorithm
Send C to all $P_k \neq P_j \in N_i$

6 Experimental Validation

To validate the performance of our algorithms we conducted experiments on a simulated network of peers. We used the accepted BRITE network generator [4] to produce a realistic network topologies containing thousands of peers and overlaid a communication tree on every such topology. For each peer, we generated s stream of random data points from a mixture of Gaussian in $\mathbb{R}^2$ and added to the data 10% uniform random noise in the range of $\mu \pm 3\sigma$. At each given time, each peer used a fixed size suffix of its stream of points as the local data. At varying intervals, we changed the means of the Gaussians, creating by that an epoch change. A typical data can be seen in Figure 2(a). The reason we chose to focus on synthetic data is the large number of factors (12) which influence the behavior of an algorithm, and the



(a) Typical dataset

(b) Typical changes in the percent of peers with $\left\|\vec{X_i}\right\| \geq \epsilon$

(c) Typical messaging throughout an experiment

Figure 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. The monitoring cost is measured on the last 80% of every epoch, in order to ignore transitional effects.

desire to perform a tightly controlled experiment in a complex algorithm which operates in an equally complex environment.

We measured the cost of the algorithm according to the frequency in which messages are sent by each peer. We report both overall cost, which includes both stationary and transitional phases of the experiment, 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 sufficient statistics computation.

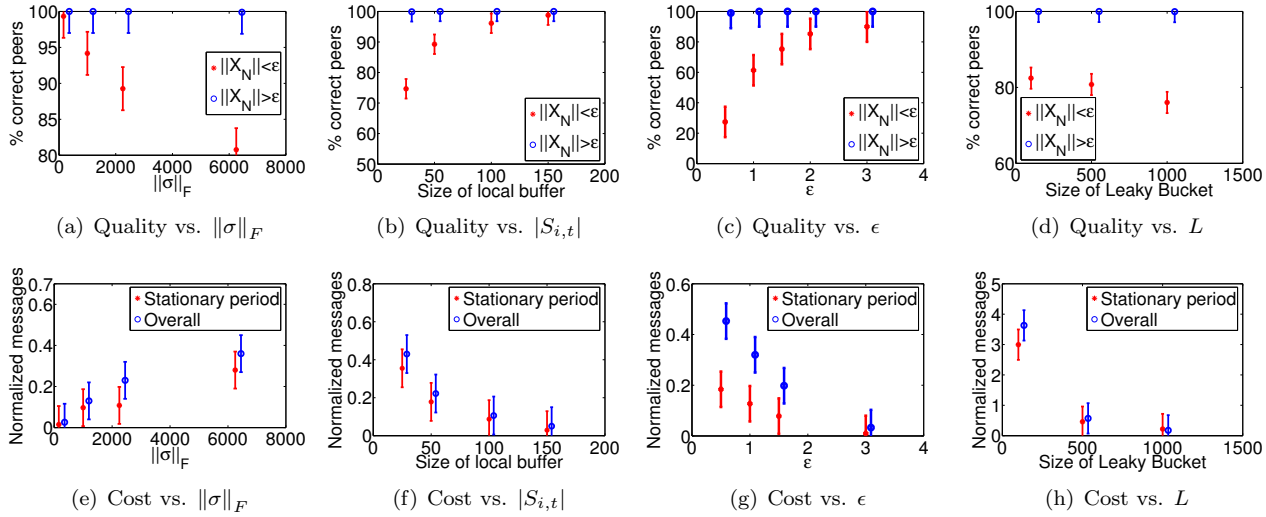
The quality of the algorithm is defined differently for the L2 Thresholding algorithm and for the Mean Monitoring and k-Means algorithms. For the L2 Thresholding algorithms, quality is measured in terms of the frequency of false positives – the percentage of peers which alert when in fact $\left\|\vec{X_N}\right\| < \epsilon$ – and false negatives – the percentage of peers which do not alert when $\left\|\vec{X_N}\right\| \geq \epsilon$. For the mean-Monitoring algorithm, quality is the average distance between $\vec{X_N}$ and the computed mean vector $\vec{\mu}$. Last, 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 the cumulation of the local data of all of the peers. Same as we do for the cost, we measure both overall quality, and quality during stationary periods.

6.1 Experiments with Local L2 Thresholding Algorithm There are many factors which may influence the performance of the L2 Thresholding algorithm. First, are those pertaining to the data: the covariance σ, and the distance between the means of the Gaussians of the different epochs (the algorithm is oblivious to the actual value 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 α and L – the maximal frequency of messages.

In our experiments, we selected the distance between the means so that the rates of false negatives and false positives are about equal. We chose an Internet topology as our topology. The rest of the parameters we scanned for various values. For each selection of the parameters we run the experiment for a long period of simulated time, allowing 10 epochs to occur. Then we measured the overall cost, and the cost of stationary periods, and the percentage of false alerts in those periods. A typical experiment is described in Figure 2.

Following, in figures 3(a) through 3(h), we explore the dependency of the average costs and quality on the Frobenius norm of the covariance of the data σ ($\|A\|_F = \sum_{i=1\ldots m}\sum_{j=1\ldots n}|a_{i,j}|^2$), the size of the local data $|S_{i,t}|$, the alert threshold ϵ, and the size of the leaky bucket L. All of these experiments were carried out with a topology of one thousand peers, and α set to 10%. In each figure, one argument is varied, and the rest remain at their default values: $|S_{i,t}| = 50$, $\epsilon = 2$, $L = 500$ (where the average edge delay is about 1100 time units), and σ of one epoch is $\begin{pmatrix} 15 & 30 \\ 30 & 15 \end{pmatrix}$ and of the other $\begin{pmatrix} 15 & -30 \\ -30 & 15 \end{pmatrix}$.

The first pair of figures, Fig. 3(a) and Fig 3(e), outline the dependency of the quality and the cost on the covariance of the data, for covariance ma-



(a) Quality vs. $\|\sigma\|_F$ (b) Quality vs. $|S_{i,t}|$ (c) Quality vs. ϵ (d) Quality vs. L

(e) Cost vs. $\|\sigma\|_F$ (f) Cost vs. $|S_{i,t}|$ (g) Cost vs. ϵ (h) Cost vs. L

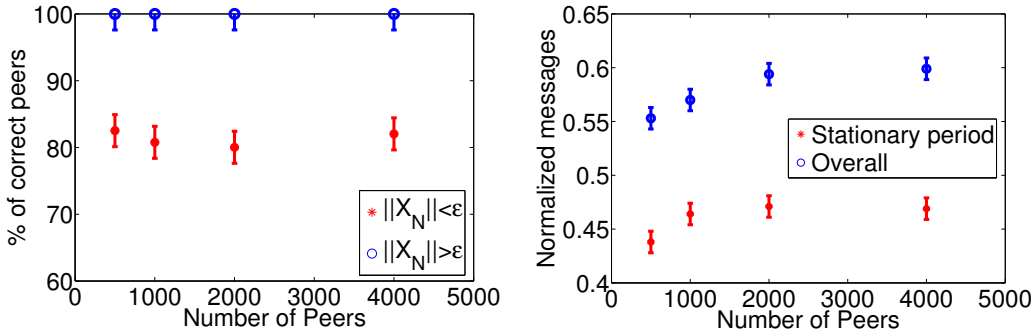
Figure 3: Dependency of cost and quality of L2 Thresholding on $\|\sigma\|_F$, $|S_{i,t}|$, ϵ, and L. Quality is defined by the percentage of peers correctly computing an alert (separated for epochs with $\left\|\vec{X_N}\right\|$ less and more than ϵ). Cost is defined as the portion of the leaky buckets intervals that are used. Both overall cost and cost of just the stationary periods are reported.

trices $\begin{pmatrix} 5 & 10 \\ 10 & 5 \end{pmatrix}$, $\begin{pmatrix} 10 & 20 \\ 20 & 10 \end{pmatrix}$, $\begin{pmatrix} 15 & 30 \\ 30 & 15 \end{pmatrix}$, and $\begin{pmatrix} 25 & 50 \\ 50 & 25 \end{pmatrix}$ (i.e., Frobenius norm of . For epochs with $\left\|\vec{X_N}\right\| < \epsilon$ both the maximal quality the average, and the worst of every experiment decrease linearly with the variance (from around 98% on average to around 82%). Epochs with $\left\|\vec{X_N}\right\| > \epsilon$, on the other hand, retained very high quality, regardless of the level of variance. As for the cost of the algorithm, this increases as the square root of $\|\sigma\|_F$ (i.e., linear to the variance), regardless of whether transitional phases are taken into account. Nevertheless, even with the highest variance, the cost stayed far from the theoretical maximum of two messages per peer per leaky bucket period (one for each neighbor, on an average of two neighbors per peer). The second pair of figures, Fig. 3(b) and Fig. 3(f), show that the variance can easily be controlled by increasing the local data. As $|S_{i,t}|$ increases, the quality increases, and cost decreases, proportionally to $\sqrt{|S_{i,t}|}$. 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.

The third pair of figures, Fig. 3(c) and Fig. 3(g), investigate the effect of requiring and ever tighter limitation on $\left\|\vec{X_N}\right\|$ by decreasing ϵ. As can be seen, beyond a given point – around $\epsilon = 2$ – the number of false positives grows drastically. The number of false negatives, on the other hand, remains constant regardless of ϵ. The cost of the algorithm decreases linearly as ϵ grows from 0.5 to 1.5, and reaches nearly zero for $\epsilon = 3$. Finally, Fig. 3(d) and Fig. 3(h) 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 and 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.

To conclude the analysis of the L2 Thresholding algorithm we investigate its scalability with the number of peers. As Figure 4 shows, the average quality and cost of the algorithm converge to a constant as the number of peers increases. This typifies local algorithms – because computation is local, the total number of peers does not affect it. Hence, there could be no deterioration in performance with scale.

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 infinitely scalable because performance does not depend on the number of peers.



(a) Average quality vs. number of peers

(b) Average cost vs. number of peers

Figure 4: Typically to a local algorithm, the average quality and cost converge to a constant when the number of peers is scaled up.

6.2 Experiments with Means-Monitoring Having explored the effects of the different parameters on the L2 Thresholding algorithm, we focus our experiments with Mean-Monitoring on three parameters that affects Mean-Monitoring behavior: τ – the alert mitigation period – and T – the length of an epoch, and on ϵ. On the cost side we also include the number of convergecast rounds which were performed.

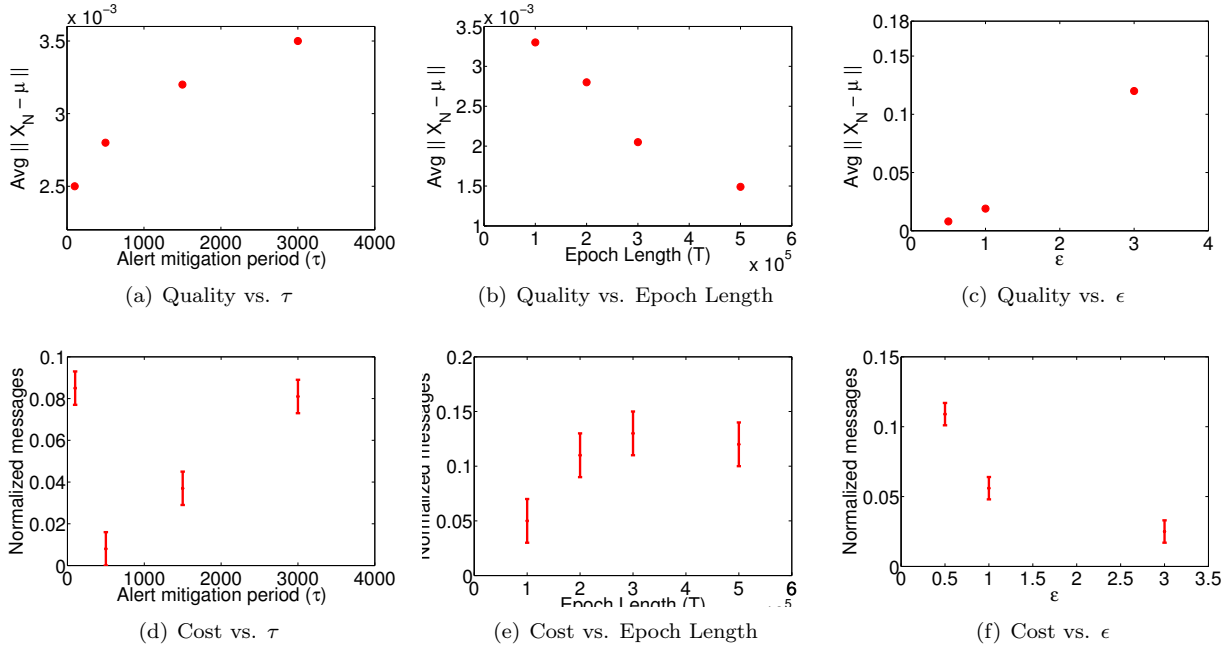
Figure 5 summarizes the results of these experiments. As can be seen the quality, measured by the distance of the actual means vector $\vec{X_N}$ from the computed one $\vec{\mu}$ is extremely good, regardless of τ or ϵ. Figure 5(b), and observations from the experiments explain why: Once the effects of the transition are over, the computed mean $\vec{\mu}$ is extremely accurate. For longer epochs those effects are amortized on a longer stationary period and the the average error becomes smaller.

As for the cost of Mean-Monitoring, except for a clear dependency on ϵ (5(f)), they show no clear trend. Our hypothesis is that because the average costs are so low, the main factor influencing them is the actual value that is sampled for $\vec{\mu}$, which can vary slightly. Note that these figures only report messages related to the L2-Thresholding algorithm which is used for monitoring the means. The actual collection of statistics was carried out between once and twice per epoch change and required two messages each times – which is negligible comparing to the messages exchanged by L2-Thresholding.

6.3 Experiments with k-Means In this set of experiments we investigate the effects of varying the remaining parameter – the sample size. To this purpose, we compare the results of our algorithm to those of a centralized algorithm which is given the same data. We compute the distance between each centroid computed by the peer-to-peer algorithm and the closest centroid computed by the centralized one. For comparison, we look at the error, vis-a-vis the output of that centralized algorithm, of a centralized algorithm which, like the distributed one, takes a sample from the entire data.

As can be seen in Fig. 6(a), the results of k-Means on a sample from centralized data and of the distributed version are equivalent. For very small samples, e.g., 500 points, the result of both algorithms are outside the prescribed ϵ threshold, which is set to 2 in this experiment. This is because k-Means output is not robust for such small samples. However, for larger samples, the quality of the distributed algorithm is within the required limit. It is interesting to note that when looking at the the accuracy throughout the execution (and not just in stationary periods) both the distributed and the centralized algorithms present large variance in the error. We conclude that this is due not to the inherent delays in the distributed algorithm. Rather, it is due to fluctuation of the output of 2-Means when, during transitional periods, the data contains a mixture from four different Gaussian (two for each epoch).

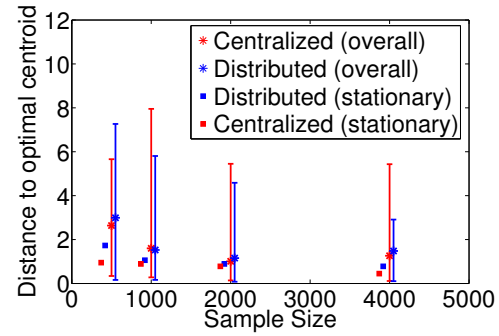
The costs of k-means have to be separated to those related to monitoring the current centroids and those related to the collection of the sample. Fig. 6(b) presents the costs of monitoring a single centroid. These should be multiplied in k to arrive at the total costs. These costs are negligible for those sample sizes that provide an output which is sufficiently accurate. We do not include here the comparative costs of centralizing the data. However, it is clear that if every change in the data is sent to the central post, these would be impermissibly large.



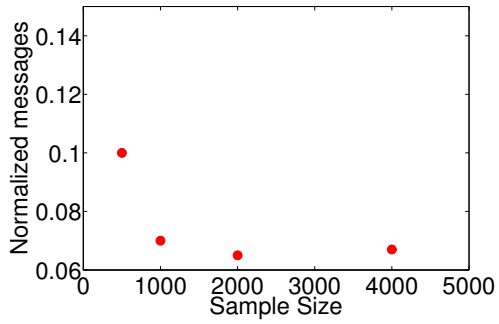
(a) Quality vs. τ (b) Quality vs. Epoch Length (c) Quality vs. ϵ

(d) Cost vs. τ (e) Cost vs. Epoch Length (f) Cost vs. ϵ

Figure 5: Dependency of cost and quality of Mean-Monitoring on different arguments



(a) Average Quality vs. Sample Size



(b) Average Monitoring Cost vs. Sample Size

Figure 6: The Quality and Cost of peer-to-peer k-means.

7 Conclusions

We presented a local algorithm which efficiently computes whether the L2 norm of the average of input vectors is above or below a given threshold. The algorithm is suitable for scenarios in which those vectors are distributed across a large peer-to-peer network, and seamlessly handles changes in the data and fail-stop failures during its execution. We then described how this algorithm can be used as the feedback loop for the control of either simple statistics or complex data mining models. This is demonstrated by describing a k-means algorithm which uses local L2 Thresholding as a feedback loop. Experiments with the k-means algorithm have shown that it can achieve extremely good accuracy with costs that, on stationary periods, are negligible. During transitional periods, the reliance on feedback allows the algorithm to make do with small samples and with a "best-effort" sampling method.

Beside the direct contributions of this work with respect to the calculation of L2 norm and of k-means in peer-to-peer networks, it also suggests an entirely new way by which local algorithms can be used in data mining. Previously, developing a local algorithm for the direct computation of a data mining model has proved difficult. However, a small number of those local algorithms is arguably sufficient to provide tools for deciding whether a data mining model correctly represents the data. Models can therefore be computed by any method (exact, approximate, or entirely random) and

then efficiently judged by the local algorithm. Furthermore, whatever those costs are for computing the model, they can be amortized on periods where the data does not change and only the local algorithm operates.

Acknowledgments

This work was supported by the United States National Science Foundation Grant IIS-0093353.

References

[1] S. Banyopadhyay, C. Giannella, U. Maulik, H. Kargupta, S. Datta, and K. Liu. Clustering distributed data streams in peer-to-peer environments. *Information Science*, in press.

[2] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. In *Proc. of SIGMOD*, 2004.

[3] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip algorithms: Design, analysis and applications. In *Proc. IEEE Infocom'05*, 2005.

[4] BRITE: Boston university Representative Internet Topology gEnerator. `http://www.cs.bu.edu/brite/`.

[5] D. Cheung, J. Han, V. Ng, and C. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proc. of ICDE*, 1996.

[6] J. Clemente, X. Défago, and K. Satou. Asynchronous peer-to-peer communication for failure resilient distributed genetic algorithms. In *Proc. of PDCS*, 2003.

[7] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie. From epidemics to distributed computing. *IEEE computer*, 37(5):60–67, May 2004.

[8] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proc. of SIGKDD*, 2001.

[9] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proc. of WWW*, 2003.

[10] D. Kempe, A. Dobra, and J. Gehrke. Computing aggregate information using gossip. In *Proc. of FoCS*, 2003.

[11] W. Kowalczyk, M. Jelasity, and A. Eiben. Towards data mining in large and fully distributed peer-to-peer overlay networks. In *Proc. of BNAIC*, 2003.

[12] D. Krivitski, A. Schuster, and R. Wolff. A local facility location algorithm for sensor networks. In *Proc. of DCOSS*, 2005.

[13] S. Kullback. *Information theory and statistics*. Dover, New York, 1968.

[14] S. Kutten and D. Peleg. Fault-local distributed mending. In *Proc. of PODC*, 1995.

[15] N. Linial. Locality in distributed graph algorithms. *SIAM Journal of Computing*, 21:193–201, 1992.

[16] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[17] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An efficient algorithm for the incremental updation of association rules in large databases. In *Proc. of SIGKDD*, 1997.

[18] R. Wolff and A. Schuster. Association rule mining in peer-to-peer systems. In *Proc. of ICDM*, 2003.

A Proof of Lemma

Proof of Lemma 2.1. By induction. For a given graph, we remove a vertex v by choosing a neighbor u and unifying them into a meta vertex w such that $S_{w,t} \leftarrow S_{u,t} \cup S_{v,t}$ and hence $|S_{w,t}| = |S_{v,t}| + |S_{u,t}|$ and $\vec{S_{w,t}} = \frac{|S_{v,t}|}{|S_{w,t}|}\vec{S_{v,t}} + \frac{|S_{u,t}|}{|S_{w,t}|}\vec{S_{u,t}}$. Then, we connect the neighbors of both u and v to w such that $N_w \leftarrow N_v \cup N_u \setminus \{u, v\}$ and for every $P_j \neq u \in N_v$ ($P_i \neq v \in N_u$) $\vec{X_{j,w}} \leftarrow \vec{X_{j,v}}$, $\vec{X_{w,j}} \leftarrow \vec{X_{v,j}}$, $\omega_{j,w} \leftarrow \omega_{j,v}$, and $\omega_{w,j} \leftarrow \omega_{v,j}$ ($\vec{X_{j,w}} \leftarrow \vec{X_{j,u}}$, $\vec{X_{w,j}} \leftarrow \vec{X_{u,j}}$, $\omega_{j,w} \leftarrow \omega_{j,u}$, and $\omega_{w,j} \leftarrow \omega_{u,j}$). Obviously, every neighbor u and v had, other than each other, has $\vec{X_{w\cap j}}$, $\omega_{w\cap j}$ equal to the ones it had with the previous neighbor (v or u) and hence that $\vec{X_{w\cap j}}$ is in A. It is left to show that $\vec{X_{w\setminus j}}$ is also in A. We look at a specific $\vec{X_{w\setminus j}}$ which connects, w.l.g., a prior neighbor of u to w. $\vec{X_{w\setminus j}} = \frac{\omega_{u\setminus j}}{\omega_{w\setminus j}}\vec{X_{u\setminus j}} + \frac{\omega_{v\setminus u}}{\omega_{w\setminus j}}\vec{X_{v\setminus u}}$. Since $\omega_{w\setminus j} = \omega_{u\setminus j} + \omega_{v\setminus u}$, $\vec{X_{w\setminus j}}$ is the weighted average of $\vec{X_{u\setminus j}}$ and $\vec{X_{v\setminus u}}$. Since A is convex and (assuming none of the weights is zero) both $\vec{X_{u\setminus j}}$ and $\vec{X_{v\setminus u}}$ are in A their weighted average is in A as well. If one of the weights is zero, then the weighted average only takes into account the other vector. If both are zero then $\omega_{w\setminus j}$ is zero as well. Hence, the resulting graph with w also maintains that all $\vec{X_{i\setminus j}}$ and $\vec{X_{i\cap j}}$ are in A, or has zero weights. Furthermore, since $\vec{X_i}$ is nothing more that the weighted average of $\vec{X_{i\setminus j}}$ and $\vec{X_{i\cap j}}$, it is also in A.

Proof of Lemma 2.3. Assume not, and that the peers have reached termination with $\left\|\vec{X_N}\right\| > \epsilon$. Then there must be two neighbors P_i and P_j such that $\hat{uf_i} \neq \hat{uf_j}$. Assume, w.l.g., $\hat{uf_i}$ is prior to $\hat{uf_j}$. This means for P_i $\hat{uf_i} \cdot \vec{X_{i\cap j}} \geq \epsilon$ while for P_j $\hat{uf_i} \cdot \vec{X_{j\cap i}} < \epsilon$ – or it would choose $\hat{uf_i}$ over $\hat{uf_j}$. However, if no more messages are traveling from P_i to P_j or vice-versa, then they must have $\vec{X_{i\cap j}} = \vec{X_{j\cap i}}$. Hence, on any termination state with $\left\|\vec{X_N}\right\| > \epsilon$, and for all P_i and $P_j \in N_i$ necessarily $\hat{uf_i} = \hat{uf_j}$.