A Local Scalable Distributed Expectation Maximization Algorithm for Large Peer-to-Peer Networks

Kanishka Bhaduri
Mission Critical Technologies Inc
Intelligent Systems Division
NASA Ames Research Center
Moffett Field, CA 94035
Kanishka.Bhaduri-1@nasa.gov

Ashok N. Srivastava
Intelligent Systems Division
NASA Ames Research Center
Moffett Field, CA 94035
Ashok.N.Srivastava@nasa.gov

ABSTRACT

This paper offers a **local** distributed algorithm for expectation maximization in large peer-to-peer environments. The algorithm can be used for a variety of well-known data mining tasks in a distributed environment such as clustering, anomaly detection, target tracking to name a few. This technology is crucial for many emerging peer-to-peer applications for bioinformatics, astronomy, social networking, sensor networks and web mining. Centralizing all or some of the data for building global models is impractical in such peer-to-peer environments because of the large number of data sources, the asynchronous nature of the peer-to-peer networks, and dynamic nature of the data/network. The distributed algorithm we have developed in this paper is provably-correct *i.e.* it converges to the same result compared to a similar centralized algorithm and can automatically adapt to changes to the data and the network. We show that the communication overhead of the algorithm is very low due to its local nature. This monitoring algorithm is then used as a feedback loop to sample data from the network and rebuild the model when it is outdated. We present thorough experimental results to verify our theoretical claims.

Categories and Subject Descriptors

H.2.4 [**Database Management Systems**]: Distributed databases; H.2.8 [**Database Applications**]: Data mining; G.3 [**Probability And Statistics**]: Multivariate statistics; C.2.4 [**Distributed Systems**]

General Terms

Algorithms, Performance, Experiments

Keywords

peer-to-peer, distributed data mining, local algorithms, EM, GMM

KDD '09 Paris, France

1. INTRODUCTION

Expectation Maximization (EM) is a powerful statistical and data mining tool which can be widely used for a variety of tasks such as clustering, estimating parameters from the data even in the presence of hidden variables, anomaly detection, target tracking and more. In 1977, Dempster *et al.* [10] presented the seminal work on EM and its application for estimating the parameters of a Gaussian Mixture Model (GMM). The authors showed that given a sample of data, there is a two step process which can estimate certain unknown parameters of the data in the presence of hidden variables. This is done by maximizing the log likelihood score of the data and assuming a generative model. Thus the classical EM algorithm is well-understood and produces satisfactory estimates of the parameters when the data is centralized.

However, there exist emerging technologies in which the data is not located at a central location but rather distributed across a large network of nodes or machines connected by an underlying communication infrastructure. The next generation Peer-to-Peer (P2P) networks such as Gnutella, BitTorrents, e-Mule, Kazaa, and Freenet offer some examples. As argued on several occasions, P2P networks can no longer be viewed as an isolated medium of data storage and dissemination; recent research on P2P web community formation [24] [9], bioinformatics[1] and diagnostics [13][32] has shown that interesting information can be extracted from the data in such networks. However data analysis in such environments calls for a new breed of algorithms which are asynchronous, highly communication efficient and scalable.

To solve this problem, in this paper we develop a local, P2P distributed (**PeDEM**) and asynchronous algorithm for monitoring and subsequent reactive updating of a GMM model using an EM style update technique. Our algorithm is provably correct *i.e.* given all the data, our algorithm will converge to the same result produced by a similar centralized algorithm. The algorithmic framework is *local*, in the sense that the computation and communication load at each node is independent of the size or the number of nodes of the network. This guarantees high scalability of the algorithm possibly to millions of nodes. The proposed methodology takes a two-step approach for building and maintaining GMM parameters in P2P networks. The first step is the *monitoring phase* in which, given an arbitrary estimate of the GMM parameters, our local asynchronous algorithm checks if they are valid with respect to the current data to within user-

[1] `http://smweb.bcgsc.bc.ca/chinook/index.html`

specified thresholds using different metrics, as we explain later. If not, this algorithm raises a flag, thereby indicating that the parameters are out-of date. At this point, we employ a convergecast-broadcast technique to rebuild the model parameters. This step is known as the *computation phase*. The correctness of the monitoring algorithm guarantees that a peer need not do anything if the flag is not raised — thus reducing communication and computation costs. When the data undergoes a change in the underlying distribution and the GMM parameters no longer represent it, the feedback loop indicates this and the parameters are recomputed. The specific contributions of this paper are as follows:

- To the best of the authors' knowledge this is one of the first attempts on developing a completely asynchronous and local algorithm for monitoring the GMM parameters in P2P networks using an EM style of update rule. Previous research has only performed computation of the parameters in a distributed setup.
- Besides this direct contribution, this paper shows how second order statistics can be directly monitored in a p2p network. Previous work [36] only showed how we can monitor first order statistics such as the mean of the data.

The rest of the paper is organized as follows. Section 2 presents the motivation of this work. Related background material is presented in Section 3. Section 4 provides some necessary background material on the EM algorithm and then introduces the notations and problem definition. Section 5 discusses the main theorem and its application for developing the algorithm. The monitoring of the parameters are presented in Section 6. Section 7 discusses the computation problem. Theoretical analysis of the algorithm is presented in Section 8 followed by experimental results in Section 9. Finally, Section 10 concludes this paper.

2. MOTIVATION

Monitoring models in large distributed environments can be done in three different ways: (1) periodic, (2) incremental, or (3) reactive. In the periodic mode, a model is build at fixed intervals of time. While this approach is simple, one needs to come up with optimal value of the interval. Too small a value of the interval may unnecessarily build models when not needed, thereby wasting resources (*e.g.* when data is stationary) while a longer interval may not update the model often in case of dynamic data. The incremental approach adjusts to the model whenever the data changes, thereby keeping the model up-to-date. However, developing incremental algorithms may be difficult. The third approach, what we take in this paper and also shown in [4][36][8], is to update a model only when the data no longer fits it. If the data is piecewise stationary, it has been shown that this approach may be both simple and efficient.

The suitability of the model and the data can be checked using several metrics: L2-norm, χ^2, log-likelihood etc. In this paper we have developed a local distributed algorithm for monitoring the GMM parameters using (1) log-likelihood of the data, and (2) norm difference between the parameters. Local algorithms rely on a set of data dependent rules, thereby deciding when a peer can stop sending messages and output the result even it has communicated with only a handful of immediate neighbors. A peer can do nothing even if its data has changed as long as its local rules are satisfied.

Practical scenarios in which distributed EM can be used

3. RELATED WORK

Work related to this research can be subdivided into two major areas — distributed EM algorithms and computation in large distributed systems a.k.a P2P systems. We provide a brief exposure to each of the topics in the next two subsections.

3.1 Distributed EM Algorithms

In standard EM algorithm, the task is to estimate some unknown parameters from the given data in the presence of some *unobserved* or *hidden* variables. The seminal work by Dempster *et al.* [10] proposed an iterative technique alternating between the **E**-step and **M**-step that solves this estimation problem. The paper also proved the convergence of the EM algorithm. In the **E**-step, the expected value of assumed hidden variables are generated using an estimate of the unknown parameters and the data. In the **M**-step, the log-likelihood of the unknown parameters given the data and hidden variables (as found in the previous step) are maximized. This two-step process is repeated until the parameter estimates converge. Furthermore, for gaussian mixture models (GMM), the updates for the M-step can be written in a closed form as a computation of the weighted combination of all data points. This decomposable nature of the problem makes it tractable in a distributed setup.

In the distributed setup, we need to find decentralized implementations of the E-step and M-step. Distributed implementation of the E-step is straightforward and communication-free: given the estimates of the parameters at each node, simply evaluate the estimates of the hidden variables based on only its local data. So the focus of most distributed EM research is to efficiently compute the parameters in the M-step in a distributed fashion. The naive approach of simply aggregating all the data at a central location does not scale well for large asynchronous networks. Next, we discuss several implementations of the distributed M-step.

In 2003, Nowak [31] proposed a distributed EM (DEM) algorithm with the execution of the M-step as follows. A ring topology is overlaid over the original network encompassing all the nodes in the network. Due to this ring, the updates in the M-step proceed in a cyclical fashion whereby at iteration t, node m gets the estimates of the parameters from its neighbor in the ring, updates those estimates with its local data, and passes them to the next neighbor at clock tick $t+1$. The paper proves that DEM monotonically converges to a local maxima and because of the incremental update, it converges more rapidly than the standard EM algorithm. However, this technique is not likely to scale for large asynchronous networks due to the strict requirement of the overlay ring topology covering all the nodes in the network. Moreover, the algorithm is highly synchronized, with each round of computation taking time which is proportional to the size of the network. This becomes problematic especially if the data changes or a node fails or joins whence the entire computation needs to be started from scratch.

To overcome this problem, several techniques have been developed. Recalling that for GMM, the updates for the M-

step can be written as weighted averages over all the nodes' data, any distributed averaging technique can be used for performing this computing over a large number of nodes. In the literature there are two basic types of distributed averaging techniques: (1) *probabilistic i.e.* gossip style — [17][6][20] in which a node repeatedly selects another node in the network and averages its data with the selected node and (2) *deterministic i.e.* graph-laplacian [26] or linear dynamical systems based [34] — in which a node repeatedly communicates with its immediate neighbors only and updates its state with the information it gets from all its neighbors. While the first class of algorithms probabilistically guarantee the correct result, the deterministic algorithms converge to the correct result asymptotically. Newscast EM [21] is the algorithm proposed by Kowalczyk and Vlassis which uses gossip-style distributed computation to compute the parameters of the M-step. At each iteration, any peer P_i selects another peer P_j at random and both compute the average of their data. It can be shown that, if the peer selection is done uniformly at random, any such gossip-based algorithm converges to the correct result exponentially fast. For such a technique to work in practice, the network must be fully connected *i.e.* any node must be able to select any other node in the network. Using deterministic averaging technique, Gu [12] proposed an EM algorithm for GMM. In this algorithm peers communicate with immediate neighbors only. At any timestamp t, whenever a peer P_i gets the current estimates from all of its immediate neighbors, it updates its own estimate based on its own data and all that it has received. Then it moves to the next timestep $t+1$ and broadcasts its own estimate to its immediate neighbors. This process continues forever. However, the major criticism for both of these techniques is that they are highly synchronous and hence not scalable for large asynchronous P2P networks.

In this paper we take a different approach. Assuming a previous estimation of the parameters, we monitor if the parameters are still valid with respect to the current global data. Our algorithmic framework guarantees correct results (with respect to centralization) at a low communication cost.

Several applications for distributed EM algorithms have also been proposed in the literature. Multi-camera tracking [27], acoustic source localization in sensor networks [19], distributed multimedia indexing [30] are some of the examples.

3.2 Data Mining in Large Distributed (P2P) Systems

Based on the type of computation performed in P2P systems, this section can be subdivided into approximate algorithms and exact algorithms.

3.2.1 Approximate Algorithms

Approximate algorithms, as the name suggests, computes the approximate data mining results. The approximation can be *probabilistic* or *deterministic*.

Probabilistic algorithms use some variations of graph random walk to sample data from their own partition and that of several neighbors' and then build a model assuming that this data is representative of that of the entire set of peers. Examples for these algorithms include the P2P k-Means algorithm by Banyopadhyay *et al.* [2], the newscast model by Kowalczyk *et al.* [20], the ordinal statistics based distributed inner product identification for P2P networks by Das *et al.* [9], the gossip-based protocols by Kempe *et al.* [17] and Boyd *et al.* [6], and more.

Researchers have proposed *deterministic* approximation technique using the method of variational approximation [16][14]. Mukherjee and Kargupta [28] have proposed distributed algorithms for inferencing in wireless sensor networks. Asymptotically converging algorithms for computing simple primitives such as mean, sum etc. have also been proposed by Mehyar *et al.* [26], and by Jelasity *et al.* [15].

3.2.2 Exact Algorithms

In exact distributed algorithms, the result produced are exactly the same if all the peers were given all the data. They can further be subdivided into *convergecast* algorithms, *flooding* algorithms and *local* algorithms.

Flooding algorithms, as the name suggests, flood whatever data is available at each node. This is very expensive especially for large systems and more so when the data changes.

In *convergecast* algorithms, the computation takes place over a spanning tree and the data is sent from the leaves up the root. Algorithms such as [33] provide generic solutions — suitable for the computation of multiple functions. These algorithms are, however, extremely synchronized.

Local algorithms are a class of highly efficient algorithms developed for P2P networks. They are data dependent distributed algorithms. However, in a distributed setup data dependency means that at certain conditions peer can cease to communicate with one another and the result is *exact*. These conditions can occur after a peer has collected the statistics of just few other peers. In such cases, the overhead of every peer becomes independent of the size of the network and hence, local algorithms exceptionally suitable for P2P networks as well as for wireless sensor networks.

In the context of graph theory, local algorithms were used in the early nineties by Linial [23] and later Afek *et al.* [1]. Naor and Stockmeyer [29] asked what properties of a graph can be computed in constant time independent of the graph size. Local algorithms for P2P data mining include the majority voting and protocol developed by Wolff and Schuster [37]. Based on its variants, researchers have further proposed more complicated algorithms: facility location [22], outlier detection [7], meta-classification [25], eigen vector monitoring [8], multivariate regression [4], decision trees [5] and the generic local algorithms [36].

Communication-efficient broadcast-based algorithms have been also developed for large clusters such as the one developed by Sharfman et al. [35]. Since these algorithms rely on broadcasts as their mode of communication, the cost quickly increases with increasing system size.

4. PRELIMINARIES

In this section we present some background material necessary to understand the PeDEM algorithm that we have developed.

4.1 Expectation Maximization

EM [10] is an iterative optimization technique to estimate some unknown parameters $\mathbf{\Theta}$ given some data $\mathbf{U}$. It is also assumed that there are some *hidden* variables $\mathbf{J}$. EM algorithm iteratively alternates between two steps to maximize the posterior probability distribution of the parameters $\mathbf{\Theta}$ given $\mathbf{U}$:

- **E-Step**: estimate the *E*xpected value of $\mathbf{J}$ given $\mathbf{\Theta}$ and $\mathbf{U}$.

- **M-Step**: re-estimate Θ to *M*aximize the likelihood of $\mathbf{U}$, given the estimates of $\mathbf{J}$ found in the previous E-step.

In order to apply the above two rules for estimation, we need closed form expressions for the E- and M-steps. Fortunately, closed form expressions exist for a widely popular estimation problem *viz.* Gaussian mixture modeling (GMM). We discuss this in details next as we will use it for the rest of the paper for developing our distributed algorithm.

A multidimensional Gaussian mixture for a random vector $\overrightarrow{x} \in \mathbb{R}^d$ is defined as the weighted combination:

$$p(\overrightarrow{x}) = \sum_{s=1}^{k} \pi_s p(\overrightarrow{x}|s)$$

of k gaussian densities where the s-th density is given by

$$p(\overrightarrow{x}|s) = \frac{1}{(2\pi)^{d/2}|\mathbf{C}_s|^{1/2}} exp\left[-(\overrightarrow{x} - \overrightarrow{\mu_s})^{\mathrm{T}} {\mathbf{C}_s}^{-1} (\overrightarrow{x} - \overrightarrow{\mu_s})/2\right]$$

each parameterized by its mean vector $\overrightarrow{\mu_s} = [\mu_{s.1}\mu_{s.2}\ldots\mu_{s.d}]^{\mathrm{T}}$ and covariance matrix $\mathbf{C}_s = (x - \mu_s)(x - \mu_s)^{\mathrm{T}}$. $\pi_s = p(s)$ defines a discrete probability distribution over the k components. Given n multi-dimensional samples $\mathbf{X} = \{\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}\}$, the task is to estimate the set of parameters

$$\Theta = \{\overrightarrow{\mu_1}, \ldots, \overrightarrow{\mu_k}, \mathbf{C}_1, \ldots, \mathbf{C}_k, \pi_1, \ldots, \pi_k\}$$

by maximizing the log likelihood of the parameters given the data:

$$\mathcal{L}(\Theta|\mathbf{X}) = \log \prod_{a=1}^{n} p(\overrightarrow{x_i}|\Theta) = \sum_{a=1}^{n} \log \left(\sum_{s=1}^{k} \pi_s \mathcal{N}(\overrightarrow{x_a}; \overrightarrow{\mu_s}, \mathbf{C}_s) \right)$$

Using EM for GMM, the E-step and the M-step can be written as:

E-step (estimate the contribution of each point):

$$q_{s,a} = \frac{\pi_s \mathcal{N}(\overrightarrow{x_a}; \overrightarrow{\mu_s}, \mathbf{C}_s)}{\sum_{r=1}^{k} \pi_r \mathcal{N}(\overrightarrow{x_a}; \overrightarrow{\mu_r}, \mathbf{C}_r)} \quad (1)$$

M-step (recompute the parameters):

$$\pi_s = \frac{\sum_{a=1}^{n} q_{s,a}}{n} \quad (2)$$

$$\overrightarrow{\mu_s} = \frac{\sum_{a=1}^{n} q_{s,a} \overrightarrow{x_a}}{\sum_{a=1}^{n} q_{s,a}} \quad (3)$$

$$\mathbf{C}_s = \frac{\sum_{a=1}^{n} q_{s,a} (\overrightarrow{x_a} - \overrightarrow{\mu}_s)(\overrightarrow{x_a} - \overrightarrow{\mu}_s)^{\mathrm{T}}}{\sum_{a=1}^{n} q_{s,a}} \quad (4)$$

where $\mathcal{N}(\overrightarrow{x_a}; \overrightarrow{\mu_s}, \mathbf{C}_s)$ denotes the pdf of a normal distribution with input $\overrightarrow{x_a}$, mean $\overrightarrow{\mu_s}$ and covariance $\mathbf{C}_s$. Note that the above computation needs to be carried out for all the k Gaussian components.

In the next few sections we shift our focus to distributed computation of these parameters and discuss some assumptions and necessary background material.

4.2 Notations and Assumptions

Let $V = \{P_1, \ldots, P_p\}$ be a set of peers connected to one another via an underlying communication infrastructure such that the set of P_i's neighbors, Γ_i, is known to P_i. Each peer communicates with its immediate neighbors (one hop neighbors) only. At time t, let $\mathcal{G}$ denote a collection of data tuples which have been generated from the k gaussian densities having unknown parameters and unknown mixing probabilities. The tuples are horizontally distributed over a large (undirected) network of machines (peers). The local data of peer P_i at time t is $S_i = [\overrightarrow{x_{i,1}}, \overrightarrow{x_{i,2}}, \ldots, \overrightarrow{x_{i,m_i}}]$, where $\overrightarrow{x_{i,j}} = [x_{i,j.1}x_{i,j.2}\ldots x_{i,j.d}]^{\mathrm{T}} \in \mathbb{R}^d$. Here m_i denotes the number of data tuples at P_i and d denotes the dimensionality of the data. The **global input** at any time is the set of all inputs of all the peers and is denoted by $\mathcal{G} = \bigcup_{i=1,\ldots,n} S_i$.

Our goal is to develop a framework under which each peer (1) checks if the current parameters of the GMM are up-to-date with respect to the global (all peers') data, and (2) recomputes the models whenever deemed unfit. The network that we are dealing with can change anytime *i.e.* peers can join or leave. Moreover, the data is dynamic and is only assumed to be piecewise stationary. The proposed algorithm is designed to seamlessly adapt to network and data changes in a communication-efficient manner.

We assume that communication among neighboring peers is reliable and ordered. These assumptions can be imposed using heartbeat mechanisms or retransmissions proposed elsewhere [11][18][36][5]. Furthermore, it is assumed that data sent from P_i to P_j is never sent back to P_i. One way of ensuring this is to assume that communication takes place over a communication tree – an assumption we make here (see [36] and [5] for a discussion of how this assumption can be accommodated or, if desired, removed).

4.3 Problem Formulation in P2P Scenario

When all the data is available at a central location, the update equations for the iterative EM algorithm are given by Equations 1–4. However, in the distributed setup, all the data is not available at a central location. Therefore, for any peer P_i, the log likelihood and the update equations, can be written as:

$$\mathcal{L}(\Theta|\mathcal{G}) = \sum_{i=1}^{p} \sum_{a=1}^{m_i} \log \left(\sum_{s=1}^{k} \pi_s \mathcal{N}(\overrightarrow{x_{i,a}}; \overrightarrow{\mu_s}, \mathbf{C}_s) \right) \quad (5)$$

E-step:

$$q_{i,s,a} = \frac{\pi_s \mathcal{N}\left(\overrightarrow{x_{i,a}}; \widehat{\overrightarrow{\mu_s}}, \widehat{\mathbf{C}_s}\right)}{\sum_{r=1}^{k} \pi_r \mathcal{N}\left(\overrightarrow{x_{i,a}}; \widehat{\overrightarrow{\mu_r}}, \widehat{\mathbf{C}_r}\right)} \quad (6)$$

M-step:

$$\pi_s = \frac{\sum_{i=1}^{p} \sum_{a=1}^{m_i} q_{i,s,a}}{\sum_{i=1}^{p} m_i} \quad (7)$$

$$\overrightarrow{\mu_s} = \frac{\sum_{i=1}^{p} \sum_{a=1}^{m_i} q_{i,s,a} \overrightarrow{x_{i,a}}}{\sum_{i=1}^{p} \sum_{a=1}^{m_i} q_{i,s,a}} \quad (8)$$

$$\mathbf{C}_s = \frac{\sum_{i=1}^{p} \sum_{a=1}^{m_i} q_{i,s,a} (\overrightarrow{x_{i,a}} - \overrightarrow{\mu}_s)(\overrightarrow{x_{i,a}} - \overrightarrow{\mu}_s)^{\mathrm{T}}}{\sum_{i=1}^{p} \sum_{a=1}^{m_i} q_{i,s,a}} \quad (9)$$

Note that computation in the E-step is entirely local to a peer. However, for the log-likelihood and the M-step, a peer needs information from all the nodes in the network in order to recompute the parameters. In this paper, we consider a monitoring version of this problem: *given a time-varying data set and pre-computed initial values of these parameters (build from a centralized or sampled data) to all peers, does these parameters describe the union of all the data held by all the peers?*

In other words, we focus on a monitoring and subsequent reactive updating of the GMM parameters. Given all the data at a central location, an admissible solution to the GMM problem occurs when the estimated parameters (given by Equations 2–4) become equal to the true parameters. However, for the distributed setup, since we are considering a dynamic scenario, we relax this criteria and consider the solution to be admissible when it is within a user defined threshold ϵ of its true value. For the monitoring problem, let $\widehat{\pi_s}$, $\widehat{\vec{\mu_s}}$ and $\widehat{\mathbf{C}_s}$ denote the parameters that were calculated offline based on some past data, and disseminated to all the peers. The monitoring problem is to check if these parameters $\widehat{\pi_s}$ (1×1), $\widehat{\vec{\mu_s}}$ ($d\times 1$) and $\widehat{\mathbf{C}_s}$ ($d \times d$) are valid with respect to the current data of all the peers. We use two different metrics to perform this (1) monitor the log-likelihood of the data, and (2) monitor the parameters themselves. Below is a formal problem definition.

Problem Definition: Given a time varying dataset S_i, user defined thresholds ϵ_1, ϵ_2, and ϵ_3, and pre-computed estimates $\widehat{\Theta} = \{\widehat{\pi_s}, \widehat{\vec{\mu_s}}, \widehat{\mathbf{C}_s}, \dots\}$, for each gaussian, the monitoring problem is to check if:

- $\mathcal{L}(\widehat{\Theta}|\mathcal{G}) < \epsilon$
- $|\pi_s - \widehat{\pi_s}| < \epsilon_1$
- $\left|\left|\vec{\mu_s} - \widehat{\vec{\mu_s}}\right|\right|^2 < \epsilon_2$
- $\left|\|\mathbf{C_s}\|_F - \widehat{\mathbf{C}_s}\right| < \epsilon_3$

for every gaussian $s \in \{1, ..., k\}$, where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. In many cases, thresholding the log-likelihood of the data may be enough. However, there are situations where monitoring the parameters may prove beneficial, as we discuss later.

4.4 Monitoring Functions in P2P Environment

As a building block of PeDEM, we use an efficient, provably correct, and local algorithm for monitoring functions of average vectors in $\mathbb{R}^d$, where the vectors are distributed in a P2P network. Here we present a brief summary; interested readers are referred to [3][36] for details.

Peers communicate with one another by sending sets of points in $\mathbb{R}^d$ or statistics as defined later in this section. Let $X_{i,j}$ denote the last sets of points sent by peer P_i to P_j. Assuming reliable messaging, once a message is delivered both P_i and P_j know $X_{i,j}$ and $X_{j,i}$. Below we present definitions of several sets which are crucial to the monitoring algorithm.

DEFINITION 4.1. *The **knowledge** of P_i is the union of S_i with $X_{j,i}$ for all $P_j \in \Gamma_i$ and is denoted by $\mathcal{K}_i = S_i \cup \bigcup_{P_j \in \Gamma_i} X_{j,i}$.*

$\mathcal{K}_i$ can also be initialized using combinations of vectors defined on S_i (instead of only S_i) as we will present in the next section.

DEFINITION 4.2. *The **agreement** of P_i and any of its neighbors P_j is $\mathcal{A}_{i,j} = X_{i,j} \cup X_{j,i}$.*

DEFINITION 4.3. *The subtraction of the agreement from the knowledge is the **withheld knowledge** of P_i with respect to a neighbor P_j* i.e. $\mathcal{W}_{i,j} = \mathcal{K}_i \setminus \mathcal{A}_{i,j}$.

The next section presents a theorem which shows how we can convert this monitoring problem into a geometric problem for an efficient solution. For this we need to split the domain into convex regions since the stopping condition we describe later (Theorem 5.1) relies on this. The following definition states the properties of these convex regions.

DEFINITION 4.4. *A collection of non-overlapping convex regions $\mathcal{R}_\mathcal{F} = \{R_1, R_2, \dots, R_\ell, T\}$ is a **cover of region** $\mathbb{R}^d$, invariant with respect to a function $\mathcal{F} : \mathbb{R}^d \to \mathbb{O}$ (where $\mathbb{O}$ is an arbitrary range), if (1) every $R_i \in \mathcal{R}_\mathcal{F}$ (except T) is convex, (2) $\mathcal{F}$ is invariant in R_i* i.e., *$\forall (x, y) \in R_i, \mathcal{F}(x) = \mathcal{F}(y)$, and (3) T denotes the area of the domain, not encompassed by $\bigcup_{i=1}^{\ell} R_i$, known as the* tie *region.*

Finally, for any $\vec{x} \in \mathbb{R}^d$ we denote $\mathcal{R}_\mathcal{F}(\vec{x})$ the first region of $\mathcal{R}_\mathcal{F}$ which includes $\vec{x}$. The precise specification of the convex regions will depend on the definition of $\mathcal{F}$. Monitoring of the GMM parameters will require us to invoke three separate monitoring problems with three separate convex regions as we show in Section 6.

The goal is to monitor and compute mixture models defined on $\mathcal{G}$. Since $\mathcal{G}$ is a hypothetical quantity, not available to any peer, each peer will estimate $\mathcal{G}$ based on only the sets of vectors defined above. However, these sets can be large, thereby making communication expensive. Fortunately, under the assumption that communication takes place over a tree topology imposed on the network, it can be shown that the same sets can be represented uniquely by only two sufficient statistics which we define next.

Set Statistics: For each set, define two statistics: (1) the *average* which is the average of all the points in the respective sets (*e.g.* $\overline{S_i}$, $\overline{\mathcal{K}_i}$, $\overline{\mathcal{A}_{i,j}}$, $\overline{\mathcal{W}_{i,j}}$, $\overline{X_{i,j}}$, $\overline{X_{j,i}}$ and $\overline{\mathcal{G}}$), and (2) the *weights* of the sets denoted by $\omega(S_i)$, $\omega(X_{i,j})$, $\omega(X_{j,i})$, $\omega(\mathcal{K}_i)$, $\omega(\mathcal{A}_{i,j})$, $\omega(\mathcal{W}_{i,j})$, and $\omega(\mathcal{G})$. Each peer communicates these two statistics for each set. We can write the following expressions for the weights and the average vectors of each set:

Knowledge

- $\omega(\mathcal{K}_i) = \omega(S_i) + \sum_{P_j \in \Gamma_i} \omega(X_{j,i})$
- $\overline{\mathcal{K}_i} = \frac{\omega(S_i)}{\omega(\mathcal{K}_i)} \overline{S_i} + \sum_{P_j \in \Gamma_i} \frac{\omega(X_{j,i})}{\omega(\mathcal{K}_i)} \overline{X_{j,i}}$

Agreement

- $\omega(\mathcal{A}_{i,j}) = \omega(X_{i,j}) + \omega(X_{j,i})$
- $\overline{\mathcal{A}_{i,j}} = \frac{\omega(X_{i,j})}{\omega(\mathcal{A}_{i,j})} \overline{X_{i,j}} + \frac{\omega(X_{j,i})}{\omega(\mathcal{A}_{i,j})} \overline{X_{j,i}}$

Withheld

- $\omega(\mathcal{W}_{i,j}) = \omega(\mathcal{K}_i) - \omega(\mathcal{A}_{i,j})$.
- $\overline{\mathcal{W}_{i,j}} = \frac{\omega(\mathcal{K}_i)}{\omega(\mathcal{W}_{i,j})} \overline{\mathcal{K}_i} - \frac{\omega(\mathcal{A}_{i,j})}{\omega(\mathcal{W}_{i,j})} \overline{\mathcal{A}_{i,j}}$

Note that these computations are local to a peer. The general methodology for computing $\mathcal{F}(\overline{\mathcal{G}})$ requires us to cover the domain of $\mathcal{F}$ using non-overlapping convex regions. For the GMM, we show the convex regions that we need for monitoring the three parameters.

Our next section presents a general criteria which a peer can use to decide the correctness of the solution based on only its local vectors.

5. GLOBALLY CORRECT TERMINATION CRITERIA

The goal of the monitoring algorithm is to raise a flag whenever the estimates of the parameters are no longer valid with respect to the union of all data *i.e.* $\mathcal{G}$. The EM monitoring algorithm guarantees eventual correctness, which means that once computation terminates, each peer computes the correct result as compared to a centralized setting. The following theorem allows a peer to stop sending messages and achieve a correct termination state *i.e.* if $\mathcal{F}(\overline{\mathcal{G}}) > \epsilon$ or $< \epsilon$ solely based on $\overline{\mathcal{K}_i}$, $\overline{\mathcal{A}_{i,j}}$, and $\overline{\mathcal{W}_{i,j}}$.

THEOREM 5.1. **[Termination Criteria]** *[36] Let $P_1, \ldots, P_n$ be a set of peers connected to each other over a spanning tree $G(V, E)$. Let $\mathcal{G}$, $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, and $\mathcal{W}_{i,j}$ be as defined in the previous section. Let R be any region in $\mathcal{R}_\mathcal{F}$. If at time t no messages traverse the network, and for each P_i, $\overline{\mathcal{K}_i} \in R$ and for every $P_j \in \Gamma_i$, $\overline{\mathcal{A}_{i,j}} \in R$ and either $\overline{\mathcal{W}_{i,j}} \in R$ or $\mathcal{W}_{i,j} = \emptyset$, then $\mathcal{F}(\overline{\mathcal{G}}) \in R$.*

PROOF. We omit the proof here. Interested readers are referred to [36]. □

The above theorem allows a peer to stop the communication and output $\mathcal{F}(\overline{\mathcal{K}_i})$ which will eventually become equal to $\mathcal{F}(\overline{\mathcal{G}})$. A peer can avoid communication even if its local data changes or the network changes as long as the result of the theorem is satisfied. Indeed, if the result of the theorem holds for every peer, and all messages have been delivered, then Theorem 5.1 guarantees this is the correct solution. Otherwise, if there exists one peer P_z for which the condition does not hold, then either of the two things will happen: (1) a message will eventually be received by P_z or, (2) P_z will send a message. In either of these two cases, the knowledge $\overline{\mathcal{K}_z}$ will change thereby guaranteeing globally correct convergence.

Using this Theorem, we now proceed to monitor each of the parameters of the GMM using the distributed EM.

6. MONITORING GMM PARAMETERS

In this section we present the monitoring of the log likelihood of the data and the three parameters given in Equations 7–9.

6.1 Monitoring log likelihood

6.2 Monitoring π_s

Monitoring π_s implies thresholding the absolute difference between the current π_s (implied by the current data) and the calculated one $\widehat{\pi_s}$ is with respect to a user defined constant ϵ_1. Denoting this difference as $Err(\pi_s)$, we can write

$$\begin{aligned} Err(\pi_s) &= |\pi_s - \widehat{\pi_s}| < \epsilon_1 \\ &\Rightarrow \left| \frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}{\sum_{i=1}^{p} m_i} - \widehat{\pi_s} \right| < \epsilon_1 \\ &\Rightarrow \left| \frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} [q_{i,s,a} - \widehat{\pi_s}]}{\sum_{i=1}^{p} m_i} \right| < \epsilon_1 \end{aligned}$$

This can be monitored using the framework presented in Section 4.4. Note that the quantity $\left[\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i}[q_{i,s,a} - \widehat{\pi_s}]}{\sum_{i=1}^{p} m_i}\right]$ is the average of the estimates in the E-step $(q_{i,s,a} - \widehat{\pi_s})$ across all the peers. Each peer subtracts $\widehat{\pi_s}$ from each of its local $q_{i,s,a}$. This forms the input S_i for this monitoring problem. However, due to the presence of the modulus operator, two concurrent monitoring problems need to be run instead of just one. Let these instances be $M_1^{\pi_s}$ and $M_2^{\pi_s}$ where $M_1^{\pi_s}$ and $M_2^{\pi_s}$ are used for checking if $Err(\pi_s) < \epsilon_1$ and $-Err(\pi_s) < \epsilon_1$ respectively. Since this monitoring problem is in $\mathbb{R}$, the convex regions are subsets of the real line. Therefore, for monitoring π_s, the following initializations need to be carried out:

- $M_1^{\pi_s}.S_i = \{q_{i,s,1} - \widehat{\pi_s}, \ldots, q_{i,s,m_i} - \widehat{\pi_s}\}$
- $M_2^{\pi_s}.S_i = \{\widehat{\pi_s} - q_{i,s,1}, \ldots, \widehat{\pi_s} - q_{i,s,m_i}\}$
- $\mathcal{R}_\mathcal{F} = \{z \in \mathbb{R} : -1 \leq z < \epsilon_1\} \bigcup \{z \in \mathbb{R} : \epsilon_1 \leq z \leq 1\}$

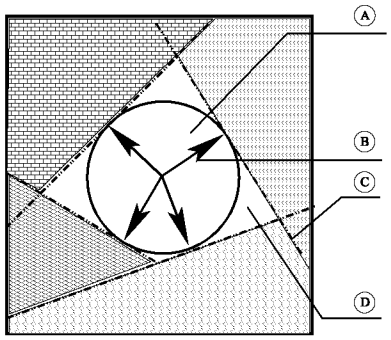


Figure 1: (A) the area inside an ϵ_2 circle (B) A random vector (C) A tangent defining a half-space (D) The areas between the circle and the union of half-spaces are the tie areas.

6.3 Monitoring $\vec{\mu_s}$

Following a similar argument, monitoring $\overrightarrow{\mu_s}$ is equivalent to thresholding the following quantity:

$$\begin{aligned} Err(\overrightarrow{\mu_s}) &= \left|\left|\overrightarrow{\mu_s} - \widehat{\vec{\mu}_s}\right|\right|^2 < \epsilon_2 \\ &\Rightarrow \left|\left| \frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}\overrightarrow{x_{i,a}}}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}} - \widehat{\vec{\mu}_s} \right|\right|^2 < \epsilon_2 \\ &\Rightarrow \left|\left| \frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}\left[\overrightarrow{x_{i,a}} - \widehat{\vec{\mu}_s}\right]}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}} \right|\right|^2 < \epsilon_2 \end{aligned}$$

The quantity $\left[\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}\left[\overrightarrow{x_{i,a}} - \widehat{\vec{\mu}_s}\right]}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}\right]$ is the average of $q_{i,s,a}\left[\overrightarrow{x_{i,a}} - \widehat{\vec{\mu}_s}\right]$ across all the peers. However the average in this case is not taken with respect to the number of tuples in the dataset S_i, but rather over all the $q_{i,s,a}$'s. As a result, we set $|S_i| = \sum_{a=1}^{m_i} q_{i,s,a}$. Moreover, for this problem, the geometric interpretation to the monitoring problem is to check if the L2-norm of the vector difference between $\overrightarrow{\mu_s} - \widehat{\vec{\mu}_s}$ lies inside a circle of radius ϵ_2. L2 norm thresholding of average data vector was first proposed in our earlier paper [36]. In $\mathbb{R}^2$, the problem can be depicted using Figure 1. The area in which $Err(\overrightarrow{\mu_s}) < \epsilon_2$, is inside the circle (sphere) and hence is already convex in $\mathbb{R}^2$ ($\mathbb{R}^d$). However, the other region outside the circle (sphere) is not convex. Hence random tangent lines (planes) are drawn on the surface of the circle (sphere) by choosing points $\widehat{u_1}, \ldots, \widehat{u_r}$ on

the circle (sphere) (the same points across all peers). Each of these half-space is convex. To check if an arbitrary point $\overrightarrow{z}$ is inside the sphere, a peers simply checks if $\|\overrightarrow{z}\| < \epsilon_2$. To check if it is outside, a peer selects the first point $\widehat{u_i}$ such that $\overrightarrow{z} \cdot \widehat{u_i} > \epsilon_2$. The following denotes the initialization necessary for this instance of the problem M^{μ_s}:

- $M^{\mu_s}.S_i = \left\{q_{i,s,1}\left(\overrightarrow{x_{i,1}} - \widehat{\overrightarrow{\mu}_s}\right), ..., q_{i,s,m_i}\left(\overrightarrow{x_{i,m_i}} - \widehat{\overrightarrow{\mu}_s}\right)\right\}$
- $M^{\mu_s}.\overline{S_i} = \frac{\sum_{a=1}^{m_i} q_{i,s,m_i}\left[\overrightarrow{x_{i,a}} - \widehat{\overrightarrow{\mu}_s}\right]}{\sum_{a=1}^{m_i} q_{i,s,a}}$, $M^{\mu_s}.\omega(S_i) = \sum_{a=1}^{m_i} q_{i,s,a}$
- $\mathcal{R}_{\mathcal{F}} = \underbrace{\{\overrightarrow{z} \in \mathbb{R}^d : ||\overrightarrow{z}|| < \epsilon_2\}}_{R_{in}} \bigcup_{i=1}^{r} \underbrace{\{\overrightarrow{z} \in \mathbb{R}^d : \overrightarrow{z} \cdot \widehat{u_i} > \epsilon_2\}}_{R_1,...,R_r}$

6.4 Monitoring C_s

The last parameter that we need to monitor is the covariance matrix $\mathbf{C}_s$. A natural extension of the L2-norm in this case is the Frobenius norm. Let $\overrightarrow{y_{i,a}} = \overrightarrow{x_{i,a}} - \overrightarrow{\mu_s}$. We have,

$$\begin{aligned}\mathbf{C}_s &= \frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}(\overrightarrow{x_{i,a}} - \overrightarrow{\mu}_s)(\overrightarrow{x_{i,a}} - \overrightarrow{\mu}_s)^{\mathrm{T}}}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}} \\ &= \frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}\begin{pmatrix} y_{i,a.1}^2 & y_{i,a.1}y_{i,a.2} & \cdots \\ & \ddots & \\ \cdots & \cdots & y_{i,a.d}^2 \end{pmatrix}}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}\end{aligned}$$

Therefore,

$$\begin{aligned}\|\mathbf{C}_s\|_F^2 &= \left(\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}y_{i,a.1}^2}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}\right)^2 \\ &+ 2\left(\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}y_{i,a.1}y_{i,a.2}}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}\right)^2 \\ &+ \cdots + \left(\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}y_{i,a.d}^2}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}\right)^2 \\ &\leq \left(\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}y_{i,a.1}^2}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}\right)^2 \\ &+ 2\left(\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}y_{i,a.1}^2}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}\right)\left(\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}y_{i,a.2}^2}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}\right) \\ &+ \cdots + \left(\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}y_{i,a.d}^2}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}\right)^2 \\ &= \left(\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}\left\{y_{i,a.1}^2 + \cdots + y_{i,a.d}^2\right\}}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}\right)^2\end{aligned}$$

By taking the square root and re-substituting $\overrightarrow{y_{i,a}}$, we get,

$$\begin{aligned}C_s^n &= \frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}\sum_{k=1}^{d}(x_{i,a.k} - \mu_{s.k})^2}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}} \\ &= \frac{\sum_{k=1}^{d}\left\{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}(x_{i,a.k} - \mu_{s.k})^2\right\}}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}} \\ &= \frac{\sum_{k=1}^{d}\left\{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}x_{i,a.k}^2 - \mu_{s.k}^2\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}\right\}}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}} \\ &= \frac{\sum_{k=1}^{d}\left\{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}x_{i,a.k}^2\right\}}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}} - \sum_{k=1}^{d}\mu_{s.k}^2 \\ &= \sum_{k=1}^{d}\left\{\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}x_{i,a.k}^2}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}} - \left(\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}x_{i,a.k}}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}\right)^2\right\}\end{aligned}$$

Thus thresholding problem is to check if $\|\mathbf{C}_s\|_F - \widehat{C_s} < \epsilon_3$ *i.e.* $\|\mathbf{C}_s\|_F < \widehat{C_s} + \epsilon_3$. As we show next, $\|\mathbf{C}_s\|_F$ is not a convex function, but C_s^n is. Thus we monitor the latter one instead. Note that, $C_s^n < \epsilon_3 \Rightarrow \|\mathbf{C}_s\|_F < \epsilon_3$. However, the other side of the inequality is not true, i.e. $C_s^n > \epsilon_3 \not\Rightarrow \|\mathbf{C}_s\|_F > \epsilon_3$. Thus using C_s^n for thresholding is more conservative: in the worst case we will have more number of false alerts for building new models, but will not miss any alert.

Let $Err(C_s^n) = C_s^n - \widehat{C_s}$. Let $g : \mathbb{R}^{2d} \to \mathbb{R}$ be defined as follows: $\forall(s_1, \ldots, s_{2d}) \in \mathbb{R}$, $g(s_1, \ldots, s_{2d}) = \sum_{i=1}^{d} s_i - \sum_{i=d+1}^{2d} s_i^2 - \widehat{C_s} - \epsilon_3$. We have the following key result:

$$Err(C_s^n) < \epsilon_3 \Leftrightarrow$$

$$g\left(\frac{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}\left(x_{i,a.1}^2, \ldots, x_{i,a.d}^2, x_{i,a.1}, \ldots, x_{i,a.d}\right)}{\sum_{i=1}^{p}\sum_{a=1}^{m_i} q_{i,s,a}}\right) < 0.$$

Each peer can locally compute the 2-d dimensional vector $q_{i,s,a}\left(x_{i,a.1}^2, \ldots, x_{i,a.d}^2, x_{i,a.1}, \ldots, x_{i,a.d}\right)$. Then the goal boils down to zero-thresholding g applied to the average of local vectors. The last thing to prove is that g (or $-g$) is a convex function. Taking the Hessian of $-g$ it can be easily shown that $-g$ is convex. We can therefore apply our tangent line technique for monitoring $Err(C_s^n)$.

Note that the inside of g is already convex. The outside can be decomposed into convex regions using tangent lines placed at random locations on g. For a 2-dimensional case, Figure 2 shows the function (a parabola) and the possible tangent lines. Let $\widehat{u_1}, \ldots, \widehat{u_z}$ be points on the parabola which define tangent lines. Checking if $g(\overline{\mathcal{K}_i}) < \epsilon_3$ is equivalent to checking if $\overline{\mathcal{K}_i}$ lies inside g. If not, we find the first point $\widehat{u_i}$ such that $\overline{\mathcal{K}_i} \cdot \widehat{u_i} > \|\widehat{u_i}\|$. We then apply the theorem for half space defined by $\widehat{u_i}$.

Now since $Err(C_s^n)$ can be both positive or negative, we need to check if $|Err(C_s^n)| < \epsilon_3$. Therefore, we need two monitoring instances denoted by $M_1^{C_s}$ and $M_2^{C_s}$. The following denotes the datasets and convex regions for this monitoring problem.

- $M_1^{C_s}.S_i = \left\{\left[q_{i,s,1}\left(x_{i,1.1}^2, \ldots, x_{i,1.d}^2, x_{i,1.1}, \ldots, x_{i,1.d}\right)\right], \ldots,\right\}$
- $M_1^{C_s}.\overline{S_i} = \frac{\sum_{a=1}^{m_i} q_{i,s,a}\left(x_{i,a.1}^2, \ldots, x_{i,a.d}^2, x_{i,a.1}, \ldots, x_{i,a.1}\right)}{\sum_{a=1}^{m_i} q_{i,s,a}}$, $M_1^{C_s}.\omega(S_i) = \sum_{a=1}^{m_i} q_{i,s,a}$
- $M_2^{C_s}.\overline{S_i} = -M_1^{C_s}.\overline{S_i}$, $M_2^{C_s}.\omega(S_i) = \sum_{a=1}^{m_i} q_{i,s,a}$
- $\mathcal{R}_{\mathcal{F}} = \left\{\overrightarrow{z} \in \mathbb{R}^{2d} : g(z) < 0\right\} \bigcup_{i=1}^{z} \{\overrightarrow{z} \in \mathbb{R}^{2d} : \overrightarrow{z} \cdot \widehat{u_i} > \|\widehat{u_i}\|\}$

6.5 Algorithm

Having discussed each of the monitoring problems, we are now in a position to present the algorithms for monitoring the parameters. For each gaussian $s \in \{1, \ldots, k\}$, we need to run the following monitoring problems separately:

- one for π_s
- one for $\overrightarrow{\mu_s}$
- one for C_s^n

In order to use Theorem 5.1 for developing a monitoring algorithm, the following steps must be followed:

1. Specify the input to the algorithm (*i.e.* S_i)

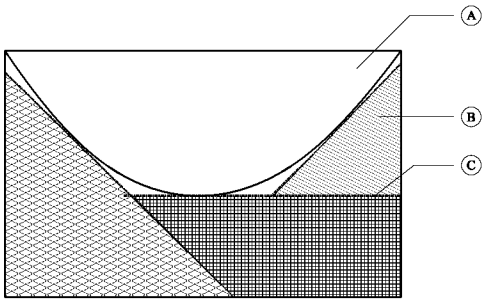


Figure 2: (A) the area inside a parabola (B) The area covered by the half-space (C) A tangent defining a half-space.

2. Specify the cover *i.e.* $\mathcal{R}_{\mathcal{F}}$

For each of the monitoring problem, these are already specified in the previous sections. Algorithm 1, 2, 3 and 4 present the pseudo-code. We describe the algorithm with respect to monitoring π_s only. The other two are almost identical.

Input: ϵ_1, $\mathcal{R}_{\mathcal{F}}$, S_i, Γ_i, L and $\widehat{\pi_s}$
Output: Set

$$flag^{\pi_s} = \begin{cases} 1 & \text{if } M_1^{\pi_s}.\overline{\mathcal{K}_i} > \epsilon_1 \bigvee M_2^{\pi_s}.\overline{\mathcal{K}_i} > \epsilon_1 \\ 0 & \text{otherwise} \end{cases}$$

Initialization:

- Generate $q_{i,s,a} \forall (\overrightarrow{x_{i,a}} \in S_i)$ using Equation 6
- Initialize two monitoring instances $M_1^{\pi_s}$ and $M_2^{\pi_s}$

On **MessageRecvd**$\left(\overline{X}, \omega(X), id\right)$ from P_j
  $M_{id}^{\pi_s}.\overline{X_{j,i}} \leftarrow \overline{X}$;
  $M_{id}^{\pi_s}.\omega\left(X_{j,i}\right) \leftarrow \omega(X)$;
  Update vectors;
On any Event:
  Call **ProcessEvent**($M_1^{\pi_s}$,$\mathcal{R}_{\mathcal{F}}$, Γ_i, L, $LastMsgSent$);
  Call **ProcessEvent**($M_2^{\pi_s}$,$\mathcal{R}_{\mathcal{F}}$, Γ_i, L, $LastMsgSent$);

Algorithm 1: Pseudo code for monitoring π_s for any peer P_i.

For any peer P_i, the input to the algorithm are ϵ, $\mathcal{R}_{\mathcal{F}}$, S_i, Γ_i, L and $\widehat{\pi_s}$ (we describe L later). The output for each of the monitoring instance is a flag which is set if the corresponding $\overline{\mathcal{K}_i}$ exceeds the threshold. In the initialization phase, it initializes its local statistics $\overline{\mathcal{K}_i}$, $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$ according to the equations in Section 4.4. The algorithm is entirely event driven. Events can be one of the following: a change in local data S_i, message received or a change in the set of neighbors Γ_i. If one of these things happen, a peer calls the **ProcessEvent** method (Algorithm 4). The goal of this method is to make sure that the conditions of Theorem 5.1 are satisfied by the peer which runs it. First peer P_i finds the active region: the region $R \in \mathcal{R}_{\mathcal{F}}$ in which $\overline{\mathcal{K}_i}$ lies *i.e.* $R = \mathcal{R}_{\mathcal{F}}(\overline{\mathcal{K}_i})$. If, $R = T$, *i.e.* the knowledge lies in the tie region, the condition of the theorem does not guarantee a solution and hence the only correct solution is flooding all of its data. On the otherhand, if for all $P_j \in \Gamma_i$, both $\overline{\mathcal{A}_{i,j}} \in R$ and $\overline{\mathcal{W}_{i,j}} \in R$, P_i does nothing and can rely on the result of the theorem for correctness. If $\overline{\mathcal{A}_{i,j}} \notin R$ or

Input: ϵ_2, $\mathcal{R}_{\mathcal{F}}$, S_i, Γ_i, L and $\widehat{\overrightarrow{\mu_s}}$
Output: Set

$$flag^{\mu_s} = \begin{cases} 1 & \text{if } \left\| M^{\mu_s}.\overline{\mathcal{K}_i} \right\| > \epsilon_2 \\ 0 & \text{otherwise} \end{cases}$$

Initialization: Initialize M^{μ_s}
On **MessageRecvd**$\left(\overline{X}, \omega(X)\right)$ from P_j
  $M^{\mu_s}.\overline{X_{j,i}} \leftarrow \overline{X}$;
  $M^{\mu_s}.\omega\left(X_{j,i}\right) \leftarrow \omega(X)$;
  Update vectors;
On any Event:
  Call **ProcessEvent**(M^{μ_s},$\mathcal{R}_{\mathcal{F}}$, Γ_i, L, $LastMsgSent$);

Algorithm 2: Pseudo code for monitoring monitoring $\vec{\mu_s}$ for any peer P_i.

$\overline{\mathcal{W}_{i,j}} \notin R$, the result of the theorem dictates P_i to send a message to P_j. Other than these two cases, a peer need not send any message even if its local data has changed.

Input: ϵ_3, $\mathcal{R}_{\mathcal{F}}$, S_i, Γ_i, L and $\widehat{C_s}$
Output: Set

$$flag^{C_s} = \begin{cases} 1 & \text{if } g\left(M_1^{C_s}.\overline{\mathcal{K}_i}\right) > 0 \bigvee g\left(M_2^{C_s}.\overline{\mathcal{K}_i}\right) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Initialization: Initialize two monitoring instances $M_1^{\sigma_s}$ and $M_2^{\sigma_s}$.
On **MessageRecvd**$\left(\overline{X}, \omega(X), id\right)$ from P_j
  $M^{\sigma_s}.\overline{X_{j,i}} \leftarrow \overline{X}$;
  $M^{\sigma_s}.\omega\left(X_{j,i}\right) \leftarrow \omega(X)$;
  Update vectors;
On any Event:
  Call **ProcessEvent**(M^{σ_s},$\mathcal{R}_{\mathcal{F}}$, Γ_i, L, $LastMsgSent$);

Algorithm 3: Pseudo code for monitoring monitoring C_s^n for any peer P_i.

Function **ProcessEvent**(M,$\mathcal{R}_{\mathcal{F}}$, Γ_i, L, $LastMsgSent$)
begin
  forall $P_j \in \Gamma_i$ **do**
    if $\mathcal{R}_{\mathcal{F}}\left(M.\overline{\mathcal{K}_i}\right) = T$ **then**
      $M.\omega\left(X_{i,j}\right) \leftarrow M.\omega\left(\mathcal{K}_i\right) - M.\omega\left(X_{j,i}\right)$;
      $M.\overline{X_{i,j}} \leftarrow \frac{M.\omega(\mathcal{K}_i) M.\overline{\mathcal{K}_i} - M.\omega\left(X_{j,i}\right) M.\overline{X_{j,i}}}{M.\omega\left(X_{j,i}\right)}$;
    end
    if $\left(M.\overline{\mathcal{A}_{i,j}} \notin \mathcal{R}_{\mathcal{F}}\left(M.\overline{\mathcal{K}_i}\right)\right)$
    $\bigvee \left(M.\overline{\mathcal{W}_{i,j}} \notin \mathcal{R}_{\mathcal{F}}\left(M.\overline{\mathcal{K}_i}\right)\right)$
    $\bigvee \left(M.\omega\left(\mathcal{W}_{i,j}\right) = 0 \bigwedge M.\overline{\mathcal{A}_{i,j}} \neq M.\overline{\mathcal{K}_i}\right)$ **then**
      Compute new $M.\omega\left(X_{j,i}\right)$ and $M.\overline{X_{j,i}}$ such that $M.\overline{\mathcal{A}_{j,i}}, M.\overline{\mathcal{W}_{j,i}} \in \mathcal{R}_{\mathcal{F}}(M.\overline{\mathcal{K}_i})$
    end
    if $CurrTime - LastMsgSent > L$ **then**
      SendMsg$(M.\overline{X_{i,j}}, M.\omega(X_{i,j}), id)$ to P_j
    end
    else Wait $(L - (CurrTime - LastMsgSent))$ units and check again
  end
end

Algorithm 4: Procedure for handling an event.

Message sending is performed in the **ProcessEvent** method

itself. When $R = T$, the peer has to flood whatever knowledge it has. Thus it sets $X_{i,j}$ and $\omega(X_{i,j})$ equals to its knowledge minus what it had received from P_j previously. It then sends this to P_j. However, when $R \neq T$, a peer can refrain from sending all data. As shown by Wolff *et al.* [36] and Bhaduri *et al.* [4], this technique of sending all the data has adverse effects on the communication in a dynamic data scenario. This is because if a peer communicates all of its data, and the data changes again later, the change is far more noisy than the original data. So we always set $\overline{X_{i,j}}$ and $|X_{i,j}|$ such that some data is retained while still maintaining the conditions of the theorem. We do this by checking with an exponentially decreasing set of values of $|\mathcal{W}_{i,j}|$ until either all $\overline{\mathcal{K}_i}$, $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}} \in R$, or $|\mathcal{W}_{i,j}|$=0. If the latter happens, then there exist no condition for which a peer can have witheld data and it has to send everything. The conditions stated in the **ProcessEvent** method are exhaustive; a peer only sends a message if one of these conditions are violated. This guarantees eventual correctness based on the theorem. Similarly, whenever it receives a message ($\overline{X}$ and $|X|$), it sets $\overline{X_{j,i}} \leftarrow \overline{X}$ and $|X_{j,i}| \leftarrow |X|$ and calls the **ProcessEvent** method again.

To prevent message explosion, in our event-based system we employ a "leaky bucket" mechanism which ensures that no two messages are sent in a period shorter than a constant L. This technique is not new but has been used earlier [4][36]. The basic idea is to maintain a timer. Whenever a peer sends a message, the timer is started. If later, the peer wants to send another message, it checks if L time units has passed since the timer was started. If yes, it sends the message and resets the timer to reflect the current time. Otherwise, the peer waits time difference between L and the 'timer' time. When the timer expires, the peer checks the conditions for sending messages and decides accordingly. Note that this mechanism does not enforce synchronization or affect correctness; at most it might delay convergence. We explore its effect thoroughly in our experiments.

In the next section we describe how we can use this monitoring algorithm to update the models, if they are outdated.

7. COMPUTING NEW MODELS

Once the models ($\widehat{\pi_s}$, $\widehat{\vec{\mu}_s}$ and $\widehat{\mathbf{C}_s}$) are monitored to within ϵ of their true values, the next step is to rebuild the models if they are found outdated. The monitoring algorithm present in the previous section generates an alert whenever one of the following occurs for any $s \in \{1...k\}$:

- $|\pi_s - \widehat{\pi_s}| \geq \epsilon_1$
- $\left|\left|\vec{\mu_s} - \widehat{\vec{\mu}_s}\right|\right|^2 \geq \epsilon_2$
- $\left|\mathbf{C}_s^n - \widehat{\mathbf{C}_s}\right| \geq \epsilon_3$

Building global models in a distributed environment is communication intensive. Here we rely on the outcome of our correct and efficient local algorithm to generate a trigger dictating the need for re-building the model. Given enough time to converge, the correctness of our monitoring algorithm ensures that even simple techniques such as best-effort sampling from the network may be sufficient to produce good results. If it does not, the underlying monitoring algorithm would eventually indicate that and a new model building will be triggered.

The idea of model computation in the network is very simple. Peers engage in a convergecast-broadcast process. The monitoring algorithm can be viewed as a flag which is raised whenever the model is misfit with respect to the global data. If this happens for any peer, it does the following. First it waits for a specific amount of time which we call the alert mitigation time τ to see if the alert is indeed due to a data change or random noise. If the alert exists even after τ units of time, the peer checks if it has received data from all its neighbors except one. If yes, it generates a sample of user-defined size B from its own data and each of its children weighing each point inversely as the size of its subtree such that each point has an equal chance of being included in the sample. It then sends the sample to its parent and marks its state as convergecast. Whenever a peer receives data from all peers, it becomes the root of the convergecast and employs a centralized EM algorithm to build new model parameters. It then sends these models to itself and marks its state to broadcast. Whenever a peer gets new models it forwards the models to all its neighbors (except the one from which it received) and moves from convergecast to broadcast phase. Because we do not impose the root of the tree, it may so happen that two peers get all the data simultaneously. We break the tie in such scenarios using the id of the nodes. Only the peer with the highest id is allowed to propagate the model in the network. Algorithm 5 presents the pseudo code of the overall EM algorithm. As shown, there are three types of messages: *Monitoring_Msg*, *Pattern_Msg* and *Dataset_Msg*. The monitoring message is passed to the underlying monitoring algorithm. The pattern message is received as part of the broadcast round while the datasets are received when the peer engages in convergeast round.

8. ANALYSIS OF ALGORITHMS

In this section we prove that (1) the PeDEM algorithm is correct, and (2) *local*.

LEMMA 8.1. *PeDEM is **eventually correct**.*

PROOF. In termination state, *i.e.* when all nodes in the network have stopped sending messages and there are no messages in transit, the knowledge $\overline{\mathcal{K}_i}$ of each peer P_i will converge to one of these states: (1) $\overline{\mathcal{K}_i} = \overline{\mathcal{G}}$, or (2) $\overline{\mathcal{K}_i}$, $\overline{\mathcal{A}_{i,j}}$, and $\overline{\mathcal{W}_{i,j}}$ are in the same $R \in \mathcal{R}_\mathcal{F}$ for every neighbor P_j. In the first case, $\overline{\mathcal{K}_i} = \overline{\mathcal{G}} \Rightarrow \mathcal{F}(\overline{\mathcal{K}_i}) = \mathcal{F}(\overline{\mathcal{G}})$. In the second case, by Theorem 5.1, $\overline{\mathcal{K}_i}$, $\overline{\mathcal{A}_{i,j}}$, and $\overline{\mathcal{W}_{i,j}} \in R \Rightarrow \overline{\mathcal{G}} \in R$. By Definition 4.4, $\mathcal{F}$ is invariant in R and hence $F(\overline{\mathcal{G}}) = F(\overline{\mathcal{K}_i})$. Thus the PeDEM algorithm is eventually correct. □

LEMMA 8.2. *PeDEM is **local**.*

9. EXPERIMENTAL RESULTS

In this section we demonstrate the performance of the PeDEM algorithm on synthetic dataset. Our implementation of the algorithm was done in Java using the DDMT[2] toolkit developed at UMBC. For the topology, we used the BRITE topology generator[3]. We experimented with the *Barabasi Albert (BA)* model since it generates realistic edge delays (in millisec), thereby simulating the internet. We convert the

[2] `http://www.umbc.edu/ddm/Sftware/DDMT/`
[3] `http://www.cs.bu.edu/brite/`

Input: ϵ_1, ϵ_2, ϵ_3, $\mathcal{R}_{\mathcal{F}}$, S_i, Γ_i, L, $\widehat{\pi_s}$, $\widehat{\overrightarrow{\mu_s}}$, $\widehat{\mathbf{C}_s}$ and τ
Output: New model such that error is less than threshold
Initialization: Initialize vectors; Set $LastDataAlert \leftarrow \infty$;$Datasent \leftarrow false$;
On Receiving a message:
$MsgType$, $Recvd\_P_j \leftarrow$ MessageRecvdFrom(P_j)
if $MsgType = Monitoring\_Msg$ **then**
  Pass Message to Monitoring Algorithm;
end
if $MsgType = Pattern\_Msg$ **then**
  Update models;
  Forward new models to all neighbors;
  $Datasent = false$;
  Restart Monitoring Algorithm with new models;
end
if $MsgType = Dataset\_Msg$ **then**
  $NumRecvd$ = Count_num_recvd();
  $Recvd\_Dataset = Recvd\_Dataset \bigcup Recvd\_P_j$;
  if $NumRecvd$=$\Gamma_i - 1$ **then**
    $flag$=Output_Monitoring_Algorithm();
    if $Datasent = false \bigwedge flag = 1$ **then**
      if $CurrTime - LastDataAlert > \tau$ **then**
        D=Sample(S_i, $Recvd\_Dataset$, B);
        $Datasent = true$;
        Send D to remaining neighbor;
      end
      else $LastDataAlert$=$CurrTime$;
      Check again in τ time;
    end
    if $flag$=0 **then** $LastDataAlert \leftarrow \infty$
  end
  if $NumRecvd$=Γ_i **then**
    D=Sample(S_i, $Recvd\_Dataset$, B);
    $NewModel$=EM(D);
    Forward $NewModel$ to all neighbors;
    $Datasent$=$false$;
    Restart Monitoring Algorithm with $NewModel$;
  end
end
if S_i, Γ_i *or* $\mathcal{K}_i$ *changes* **then**
  Run Monitoring Algorithm;
  $flag$=Output_Monitoring_Algorithm();
  if $flag$=1 *and* P_j=*IsLeaf()* **then**
    Execute the same conditions as $MsgType$=$Dataset\_Msg$;
  end
end

Algorithm 5: P2P EM Algorithm.

edge delays to simulator ticks for time measurement since wall time is meaningless when simulating thousands of computers on a single PC. On top of each network generated by BRITE, we overlay a communication tree.

9.1 Data Generation

The input data of a peer is a set of vectors in $\mathbb{R}^d$ generated according to multi-dimensional GMM. More specifically, for a given experiment, we fix the number of gaussians k, their means $\overrightarrow{\mu_1}, \ldots, \overrightarrow{\mu_k}$ and covariance matrices $\mathbf{C}_1, \ldots, \mathbf{C}_k$, and also the mixing probabilities $\pi_1, \ldots, \pi_k$. Every time a simulated peer needs an additional data point, it first selects a gaussian s with corresponding probability π_s and then generates a gaussian vector in $\mathbb{R}^d$ with mean and covariance $\overrightarrow{\mu_s}, \mathbf{C}_s$. The means and the covariances are changed randomly at controlled intervals to create an epoch change.

9.2 Measurement Metric

In our experiments, the two most important parameters for measurement are the *quality* of the result and the *cost* of the algorithm.

For the regression monitoring algorithm, quality is measured in terms of the percentage of peers which correctly compute an alert, *i.e.*, the number of peers which report that $\overline{\mathcal{K}_i} < \epsilon$ when $\mathcal{E}^{\mathcal{G}} < \epsilon$ and similarly $\overline{\mathcal{K}_i} > \epsilon$ when $\mathcal{E}^{\mathcal{G}} > \epsilon$. We also report the overall quality which is average of the qualities for both less than and greater than ϵ and hence lies in between those two. Moreover, for each quality graph in Figures **??**, **??**, **??**, **??**, **??** and **??** we report two quantities — (1) the average quality over all peers, all epochs and 10 independent trials (the center markers) and (2) the standard deviation over 10 independent trials (error bars). For the regression computation algorithm, quality is defined as the L2 norm distance between the solution of our algorithm and the actual regression weights. We compare this to a centralized algorithm having access to all of the data.

We refer to the cost of the algorithm as the number of *normalized messages* sent, which is the number of messages sent by each peer per unit of leaky bucket L. Hence, 0.1 normalized messages means that nine out of ten times the algorithm manages to avoid sending a message. We report both overall cost and the monitoring cost (stationary cost), which refers to the "wasted effort" of the algorithm. We also report, where appropriate, messages required for convergecast and broadcast of the model.

9.3 Typical Experiments

A typical experiment is shown in Figure **??**. In all the experiments, about 4% of the data of each peer is changed every 1000 simulator ticks. Moreover, after every 5×10^5 simulator ticks, the data distribution is changed. Therefore there are two levels of data change — (1) every 1000 simulator ticks we sample 4% of new data from the same distribution (stationary change) and (2) every 5×10^5 clock ticks we change the distribution (non-stationary change). To start with, every peer is supplied the same regression coefficients as the coefficients of the data generator. Figure **??** shows that for the first epoch, the quality is very high (nearly 96%). After 5×10^5 simulator ticks, we change the weights of the generator without changing the coefficients given to each peer. Therefore the percentage of peers reporting $\overline{\mathcal{K}_i} < \epsilon$ drops to 0. For the cost, Figure **??** shows that the monitoring cost is low throughout the experiment if we ignore the transitional effects.

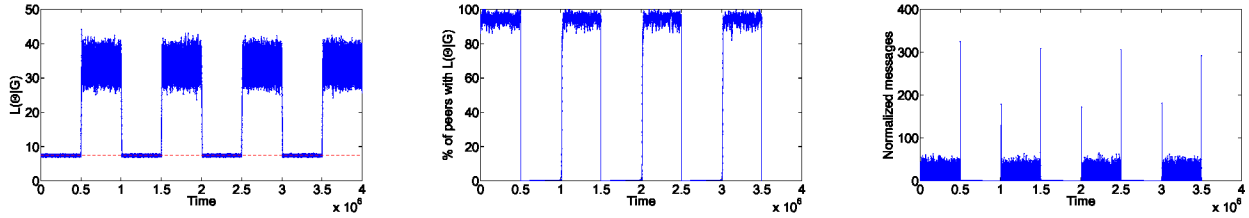
9.4 Results: Regression Monitoring

9.5 Results: Regression Models

10. CONCLUSION

Acknowledgements

This work was supported by the NASA Aviation Safety Program, Integrated Vehicle Health Management Project.

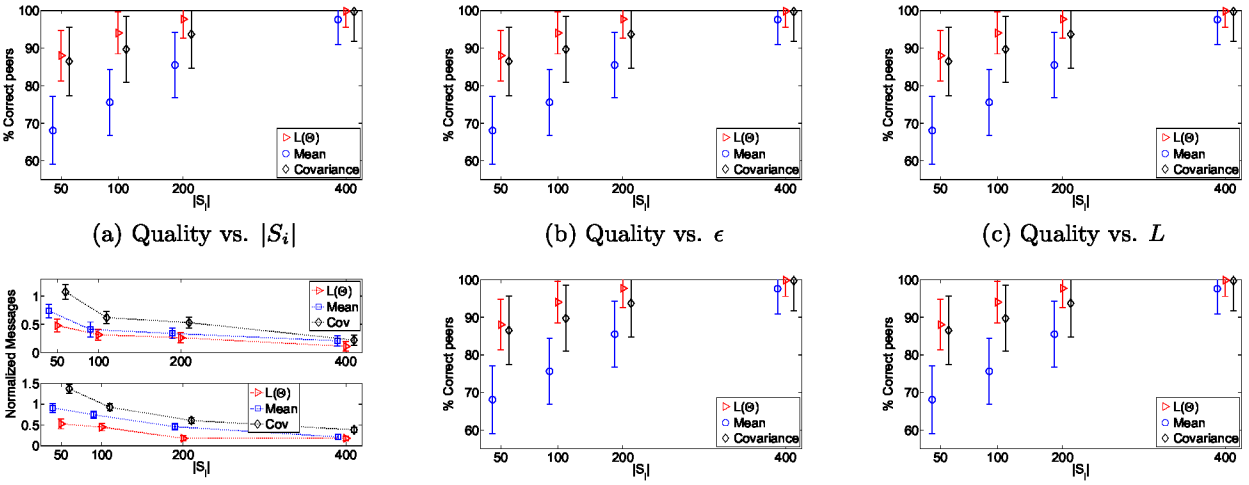


(a) Variation of $\mathcal{L}(\Theta|\mathcal{G})$ vs. time for the synthetic dataset. The red line refers to $\mathcal{L}(\Theta|\mathcal{G}) < \epsilon$. ϵ.

(b) Percentage of peers reporting

(c) Number of messages per peer per unit of L.

Figure 3: Plot of typical experiments. Each experiment is repeated for several epochs. Quality is measured both inside and outside ϵ. Cost is measured during the entire experiment and during stationary phases. Last 80% of the time refers to stationary phase to ignore transitional effects.



(a) Quality vs. $|S_i|$

(b) Quality vs. ϵ

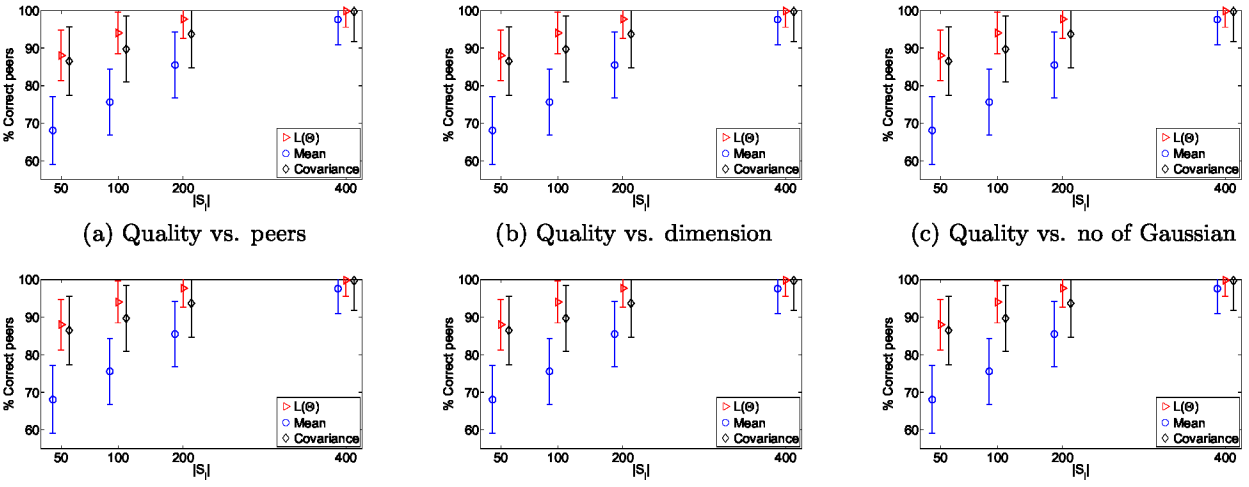
(c) Quality vs. L

(d) Cost vs. $|S_i|$

(e) Cost vs. ϵ

(f) Cost vs. L

Figure 4: Variation of the quality and cost of the monitoring algorithm on the different parameters. We have separated the quality of the log-likelihood, mean and covariance monitoring algorithm. We have also separated the cost of the entire experiment and during stationary phase.



(a) Quality vs. peers

(b) Quality vs. dimension

(c) Quality vs. no of Gaussian

(d) Cost vs. peers

(e) Cost vs. dimension

(f) Cost vs. no of Gaussian

Figure 5: Scalability of the monitoring algorithm with respect to number of peers, dimension of each gaussian, and number of gaussians.

11. REFERENCES

[1] Y. Afek, S. Kutten, and M. Yung. The Local Detection Paradigm and Its Application to Self-Stabilization. *In Theoretical Computer Science*, 186(1–2):199–229, October 1997.

[2] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, and S. Datta. Clustering Distributed Data Streams in Peer-to-Peer Environments. *Information Science*, 176(14):1952–1985, 2006.

[3] K. Bhaduri. *Efficient Local Algorithms for Distributed Data Mining in Large Scale Peer to Peer Environments: A Deterministic Approach.* PhD thesis, University of Maryland, Baltimore County, March 2008.

[4] K. Bhaduri and H. Kargupta. A Scalable Local Algorithm for Distributed Multivariate Regression. *Statistical Analysis and Data Mining*, 1(3):177–194, November 2008.

[5] K. Bhaduri, R. Wolff, C. Giannella, and H. Kargupta. Distributed Decision Tree Induction in Peer-to-Peer Systems. *Statistical Analysis and Data Mining Journal*, 1(2):85–103, June 2008.

[6] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip Algorithms: Design, Analysis and Applications. In *Proceddings Infocom'05*, pages 1653–1664, Miami, March 2005.

[7] J. Branch, B. Szymanski, C. Gionnella, R. Wolff, and H. Kargupta. In-Network Outlier Detection in Wireless Sensor Networks. In *Proceedings of ICDCS'06*, Lisbon, Portugal, July 2006.

[8] K. Das, K. Bhaduri, S. Arora, W. Griffin, K. Borne, C. Giannella, and H. Kargupta. Scalable Distributed Change Detection from Astronomy Data Streams using Local, Asynchronous Eigen Monitoring Algorithms. In *Proceedings of SDM'09 (accepted)*, Sparks, NV, 2009.

[9] 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)*, 20(4):475–488, 2008.

[10] A. Dempster, N. Laird, and D. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *J. R. Statist. Soc., Series(B)*, 39(1):1–38, 1977.

[11] J. Garcia-Luna-Aceves and S. Murthy. A Path-Finding Algorithm for Loop-Free Routing. *IEEE Transactions on Networking*, 5(1):148–160, 1997.

[12] D. Gu. Distributed EM Algorithm for Gaussian Mixtures in Sensor Networks. *IEEE Transactions on Neural Networks*, 19(7):1154–1166, July 2008.

[13] Q. Huang, H. J. Wang, and N. Borisov. Privacy-Preserving Friends Troubleshooting Network. In *Proceedings of NDSS'05*, 2005.

[14] T. Jaakkola. Tutorial on Variational Approximation Methods. *In Advanced Mean Field Methods: Theory and Practice*, 2000.

[15] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based Aggregation in Large Dynamic Networks. *ACM Transactions on Computer Systems (TOCS)*, 23(3):219–252, August 2005.

[16] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An Introduction to Variational Methods for Graphical Models. *Machine Learning*, 37(2):183–233, November 1999.

[17] D. Kempe, A. Dobra, and J. Gehrke. Computing Aggregate Information using Gossip. In *Proceedings of FOCS'03*, Cambridge, 2003.

[18] P. M. Khilar and S. Mahapatra. Heartbeat Based Fault Diagnosis for Mobile Ad-Hoc Network. In *Proceedings of IASTED'07*, pages 194–199, Phuket, Thailand, 2007.

[19] N. Kitakoga and T. Ohtsuki. Distributed EM Algorithms for Acoustic Source Localization in Sensor Networks. In *Proceedings of VTC'06*, pages 1–5, Montreal, Canada, September 2006.

[20] 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*, pages 203–210, University of Nijmegen, 2003.

[21] W. Kowalczyk and N. A. Vlassis. Newscast EM. In *Proceedings of NIPS'04*, pages 713–720, Vancover, Canada, December 2004.

[22] D. Krivitski, A. Schuster, and R. Wolff. A Local Facility Location Algorithm for Large-Scale Distributed Systems. *Journal of Grid Computing*, 5(4):361–378, 2007.

[23] N. Linial. Locality in Distributed Graph Algorithms. *SIAM Journal of Computing*, 21:193–201, 1992.

[24] K. Liu, K. Bhaduri, K. Das, P. Nguyen, and H. Kargupta. Client-side Web Mining for Community Formation in Peer-to-Peer Environments. *SIGKDD Explorations*, 8(2):11–20, 2006.

[25] P. Luo, H. Xiong, K. Lü, and Z. Shi. Distributed Classification in Peer-to-Peer Networks. In *Proceedings of SIGKDD'07*, pages 968–976, 2007.

[26] M. Mehyar, D. Spanos, J. Pongsajapan, S. H. Low, and R. Murray. Distributed Averaging on Peer-to-Peer Networks. In *Proc. of CDC'05*, Spain, 2005.

[27] T. Mensink, W. Zajdel, and B. Kröse. Distributed EM Learning for Multi-Camera Tracking. In *Proceedings of ICDSC'07*, pages 178–185, Stanford, CA, Sept 2007.

[28] S. Mukherjee and H. Kargupta. Distributed Probabilistic Inferencing in Sensor Networks using Variational Approximation. *Journal of Parallel and Distributed Computing (JPDC)*, 68(1):78–92, January 2008.

[29] M. Naor and L. Stockmeyer. What Can be Computed Locally? In *Proceedings of STOC'93*, pages 184–193, 1993.

[30] A. Nikseresht and M. Gelgon. Gossip-Based Computation of a Gaussian Mixture Model for Distributed Multimedia Indexing. *IEEE Transactions on Multimedia*, 10(3):385–392, April 2008.

[31] R. D. Nowak. Distributed EM Algorithms for Density Estimation and Clustering in Sensor Networks. *IEEE Transactions on Signal Processing*, 51(8):2245–2253, August 2003.

[32] N. Palatin, A. Leizarowitz, and A. Schuster. Mining for Misconfigured Machines in Grid Systems. In *Proceedings of KDD'06*, pages 687–692, 2006.

[33] M. Rabbat and R. Nowak. Distributed Optimization in Sensor Networks. In *Proceedings of IPSN'04*, pages 20–27, Berkeley, California, USA, 2004.

[34] D. Scherber and H. Papadopoulos. Distributed Computation of Averages Over ad hoc Networks. *IEEE Journal on Selected Areas in Communications*, 23(4):776–787, 2005.

[35] I. Sharfman, A. Schuster, and D. Keren. A Geometric Approach to Monitoring Threshold Functions over Distributed Data Streams. In *Proceedings of SIGMOD'06*, pages 301–312, Chicago, Illinois, June 2006.

[36] R. Wolff, K. Bhaduri, and H. Kargupta. A Generic Local Algorithm for Mining Data Streams in Large Distributed Systems. *IEEE Transactions on Knowledge and Data Engineering (accepted for publication)*, 2008.

[37] R. Wolff and A. Schuster. Association Rule Mining in Peer-to-Peer Systems. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 34(6):2426 – 2438, December 2004.