# A Scalable Local Algorithm for Distributed Multivariate Regression

Kanishka Bhaduri*, Hillol Kargupta †*

*Department of Computer Science and Electrical Engineering,

University of Maryland, Baltimore County,

1000 Hilltop Circle,

Baltimore, Maryland, 21250, USA

Email:{kanishk1,hillol}@cs.umbc.edu

§The author is also affiliated to Agnik, LLC., Columbia, MD, USA

A shorter version of this paper was published in SIAM Data Mining Conference 2008

**Abstract**

This paper offers a local distributed algorithm for multivariate regression in large peer-to-peer environments. The algorithm can be used for distributed inferencing, data compaction, data modeling and classification tasks in many emerging peer-to-peer applications for bioinformatics, astronomy, social networking, sensor networks and web mining. Computing a global regression model from data available at the different peer-nodes using a traditional centralized algorithm for regression can be very costly and impractical because of the large number of data sources, the asynchronous nature of the peer-to-peer networks, and dynamic nature of the data/network. This paper proposes a two-step approach to deal with this problem. First, it offers an efficient local distributed algorithm that monitors the "quality" of the current regression model. If the model is outdated, it uses this algorithm as a feedback mechanism for rebuilding the model. The local nature of the monitoring algorithm guarantees low monitoring cost. Experimental results presented in this paper strongly support the theoretical claims.

**Index Terms**

peer-to-peer, data mining, decision trees

## I. Introduction

Multivariate Regression (MR) is a powerful statistical and machine learning tool that is widely used for prediction, classification, and data compression. Multivariate regression is relatively well understood given a sample of the data (input variables and corresponding target output values) at a single location. However, there are many emerging scenarios where data is distributed over a network of machines. Peer-to-Peer (P2P) networks offer one such scenario. P2P systems such as Gnutella, BitTorrents, e-Mule, Kazaa, and Freenet are increasingly becoming popular for many applications that go beyond downloading music without paying for it. Examples include P2P systems for network storage, web caching, searching and indexing of relevant documents and distributed network-threat analysis. The next generation of advanced P2P applications for bioinformatics [1] and client-side web mining [2][3] are likely to need support for advanced data analysis and mining. Performing basic operations like regression is very challenging in a P2P network because of the large number of data sources, the asynchronous nature of the P2P networks, and dynamic nature of the data.

This paper offers a local distributed algorithm for performing multivariate regression and monitoring the model in a P2P network. The approach is scalable, decentralized, asynchronous,

and inherently based on in-network computation. 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 is very important for the scalability of the algorithm in large P2P networks. The proposed methodology takes a two-step approach for building and maintaining MR models in P2P networks. The first step in our algorithm is the *monitoring phase* in which, given an estimate of the MR model to all the peers, they asynchronously track any change between the model and the global data using a provably correct local algorithm. The second step, known as the *computation phase*, uses the monitoring algorithm as a feedback loop for triggering a new round of MR model-building if necessary. The algorithm guarantees that as long as the MR model correctly represents the data, little computing and communication resources are spent for monitoring the environment. When the data undergoes a change in the underlying distribution and the MR model no longer represents it, the feedback loop indicates this and the model is rebuilt. Moreover, we also show that all the data need not be centralized to recompute the MR coefficients. We can do in-network aggregation for finding them; thereby using far less resources than brute force centralization. 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 doing multi-variate regression in P2P networks which is robust to data and network changes.
- Besides this, we have also derived an upper bound on the total number messages exchanged between the peers in the worst case.
- Most of the previous work in the literature focuses on linear regression in distributed environments. Our technique can be applied to most types of common multivariate regression.

The rest of the paper is organized as follows. Related background material is presented in Section II. Section III introduces the notations and problem definition. Section IV presents the MR monitoring algorithm, while Section V discusses the MR computation problem. Experimental results are presented in Section VII. Finally, Section VIII concludes this paper.

## II. Background

This section provides the necessary background material.

### A. Approach

Statistical models can be built and updated from distributed data in various ways. The *periodic* approach is to simply rebuild the model from time to time. The *incremental* approach is to update the model whenever the data changes. Lastly, the *reactive* approach, what we propose here, is to monitor the change, and rebuild the model only when it no longer suits the data. The *periodic* approach can be highly inefficient since, there is the risk of wasting resources even if the data is stationary and also the risk of model inaccuracy if the updating is delayed. *Incremental* algorithms can be very efficient; however their major drawback is that a separate algorithm needs to be handcrafted for every problem. Data driven *reactive* algorithms are efficient, simple and can accommodate a wide variety of function computation. This is because the algorithm only reacts and rebuilds the model if the data changes — in other cases, the algorithm does nothing and saves unnecessary communication.

The work presented in this paper considers building and updating regression models from data distributed over a P2P network where each peer contains a subset of the data tuples. In the distributed data mining literature, this is usually called the horizontally partitioned or homogeneously distributed data scenario. Building a global regression model (defined on the union of all the data of all the peers) in large-scale networks and maintaining it is a vital task. Consider a network where there are a number of nodes (by node we mean peers, sensors, grid components etc.) and each node gets a stream of tuples (can be sensor readings, music files etc.) frequently. We may wish to build a regression model on the global data to (1) compactly represent the data and (2) predict the value of a target variable. This is difficult since the data is distributed and more so because it is dynamic. Centralization obviously does not work because the data may change at a faster rate than the rate at which it can be centralized. Local algorithms are an excellent choice in such scenarios since in a local algorithm, each peer computes the result based on the information from only a handful of nearby neighbors. Hence local algorithms are highly scalable and offer bounded communication complexity per peer. Therefore, such an algorithm will enable the user to monitor regression models using low resources.

### B. Related Work

The work presented in this paper is related to two main bodies of literature - multivariate regression and computation in large distributed environments.

*1) Distributed Multi-variate Regression:* The problem of distributed multivariate regression has been addressed by many researchers till date. Hershberger et al. [4] considered the problem of performing global MR in a vertically partitioned data distribution scenario. The authors propose a wavelet transform of the data such that, after the transformation, effect of the cross terms can be dealt with easily. The local MR models are then transported to the central site and combined to form the global MR model. Such synchronized techniques are unlikely to scale in large, asynchronous systems such as modern P2P networks.

Many researchers have looked into the problem of doing distributed MR using distributed kernel regression techniques such as Guestrin et al. [5] and Predd et al. [6]. The algorithm presented by Guestrin et al. [5] performs linear regression in a network of sensors using in-network processing of messages. Instead of transmitting the raw data, the proposed technique transmits constraints only, thereby reducing the communication complexity drastically. Similar to the work proposed here, their work also uses local rules to prune messages. However the major drawback is that their algorithm is not suitable for dynamic data. It will be very costly if the data changes since, as the authors point out, two passes are required over the entire network to make sure that the effect of the measurements of each node are propagated to every other node. Moreover, contrary to the broad class of problems that we can solve using our technique, their technique is only applicable for solving the linear regression problem.

Meta-learning is an interesting class of algorithms typically used for supervised learning. In a meta learning, such as bagging [7] or boosting [8] many models are induced from different partitions of the data and these “weak” models are combined using a second level algorithm which can be as simple as taking the average output of the models for any new sample. Such a technique is suitable for inducing models from distributed data as proposed by Stolfo et al. [9]. The basic idea is to learn a model at each site locally (no communication at all) and then, when a new sample comes, predict the output by simply taking an average of the local outputs. Xing et al. [10] present such a framework for doing regression in heterogenous datasets. However, these techniques perform poorly as the number of such data partitions increases to millions – as in typical P2P systems.

A closely related topic is classification in which the output is discrete instead of real-valued. Several algorithms have been proposed for distributed classification. Here we present only a few of them.

Caragea *et al.* [11] presented a decision tree induction algorithm for both horizontally and vertically distributed data. Noting that the crux of any decision tree algorithm is the use of an effective splitting criteria, the authors propose a method by which this criteria can be evaluated in a distributed fashion. Their system is available as part of the INDUS system. A different approach was taken by Giannella *et al.* [12] and Olsen [13] for inducing decision tree in vertically partitioned data. They used Gini information gain as the impurity measure and showed that Gini between two attributes can be formulated as a dot product between two binary vectors. To reduce the communication cost, the authors evaluated the dot product after projecting the vectors in a random smaller subspace. The major disadvantages of these techniques are (1) strong synchronization requirements and (2) inability to adapt to changes in data or network.

Distributed probabilistic classification on heterogenous data sites have also been discussed by Merugu and Ghosh [14]. Similarly, Park *et al*. have proposed a fourier spectrum-based approach for decision tree induction in vertically partitioned datasets [15].

Meta-classification from horizontally partitioned data for large distributed systems have been proposed by Lou et al. [16]. The system builds local models which requires no communication at all. When a new tuple arrives, it is broadcast to all the nodes and the output is determined using a variation of the majority voting scheme which the authors term as distributed plurality voting (DPV). Two disadvantages of this method are as follows. The tuple to be classified needs to be flooded in the network. Similar to other meta-learning techniques, the quality of such algorithms degrade as the size of the systems increases to millions of peers.

A robust, completely asynchronous and communication efficient algorithm for decision tree induction from horizontally partitioned data distributed in large P2P systems has been proposed by Bhaduri et al. [17]. The algorithm is eventually correct which means the decision tree inducted by our algorithm is the same that would be induced given all the data at a central location. The algorithm also seamlessly handles changes in the data and the network. Experimental results show the low cost of building and maintaining the decision trees even when the data changes.

*2) Computation in large distributed (P2P) systems:* Computation for P2P networks span three main areas: (1) best effort heuristics, (2) gossip based computations, (3) broadcast-based systems and (4) local algorithms. For a detailed survey interested readers can refer to [18].

Algorithms using best effort heuristics have been developed for large distributed systems. The P2P $k$-Means algorithm by Bandyopadhyay et al. [19] is one such example. Typically for such

algorithms, a peer collects some samples from its own data and its neighbors and builds a model on this sample. The samples are generally collected using some variations of random walk-based techniques. These algorithms can be classified as probabilistic approximate algorithms since the results are bounded only on average. A different class is the set of deterministic approximate algorithms such as the inferencing problem in sensor networks using variational approximation technique proposed by Mukherjee et al. [20].

Gossip algorithms rely on the properties of random samples to provide probabilistic guarantees on the accuracy of the results. Researchers have developed different approaches for performing basic operations (*e.g.* average, sum, max, random sampling) on P2P networks using gossip techniques. Kempe *et al.* [21] and Boyd *et al.* [22] present such primitives. In gossip protocols, a peer exchanges data or statistics with a random peer. However, they can still be quite costly – requiring hundreds of messages per peer for the computation of just one statistic. Another closely related technique is to use deterministic gossip or flooding. In flooding, every peer floods/broadcasts the data and therefore, eventually the data/statistic is propagated through the entire network. Here again the major drawback is scalability and the answer to dynamic data.

Communication-efficient broadcast-based algorithms have been also developed for large clusters such as the one developed by Sharfman et al. [23]. Since these algorithms rely on broadcasts as their mode of communication, the cost quickly increases with increasing system size.

Local algorithms are a good choice for data mining in P2P networks since in a local algorithm, the result is generally computed by communicating with a handful of nearby neighbors and the total communication per peer is also bounded. Local algorithms rely on data dependent conditions which we refer to as local rules, to stop propagating messages. This means that if the data distribution does not change, the communication overhead is very low. On the other hand, the local rules are violated when the distribution changes. On one hand, local algorithms are highly efficient (and hence scalable). The exact local algorithms we consider in this paper guarantee eventual convergence to the *exact* result (equal to that which would be computed given the entire data). This feature makes local algorithms exceptionally suitable for P2P networks as well as to wireless sensor networks.

The idea of using local rules for algorithms dates back to the seventies. John Holland described such rules for non-linear adaptive systems and genetic algorithms in his seminal work for biological systems [24]. Local evolutionary rules for grid-based cellular automaton were first

introduced in 1950's by John Von Neumann [25] and later adopted in many fields such as artificial agents, VLSI testing, physical simulations to mention a few. In the context of graph theory, local algorithms were used in the early nineties by Afek *et al.* [26] and Linial [27]. Naor and Stockmeyer [28] asked what properties of a graph can be computed in constant time independent of the graph size. Kutten and Peleg [29] have introduced local algorithms for fault-detection in which the cost depends only on the unknown number of faults and not on the entire graph size. They have developed solutions for some key problems such as the maximal independent set (MIS) and graph coloring. Kuhn *et al.* [30] have suggested that some properties of graphs cannot be computed locally.

More recently, local algorithms have been developed for several data mining problems: association rule mining [31], facility location [32], L2 Thresholding [33], outliers detection [34], meta-classification [16] and decision tree induction [17]. Researchers have also looked at the complexity of local algorithms using the concept of veracity radius [35].

## III. Notations and Problem Definition

### A. Notations

Let $V = \{P_1, \ldots, P_n\}$ be a set of peers connected to one another via an underlying communication infrastructure such that the set of $P_i$'s neighbors, $\Gamma_i$, is known to $P_i$. Additionally, at a time $t$, $P_i$ is given a stream of data vectors in $\mathbb{R}^d$. The local data of peer $P_i$ at time $t$ is $S_i = \left[\left(\overrightarrow{x_1^i}, f(\overrightarrow{x_1^i})\right), \left(\overrightarrow{x_2^i}, f(\overrightarrow{x_2^i})\right), \ldots\right]$, where each $\overrightarrow{x_j^i}$ is a ($d$-1)-dimensional data point $\left[x_{j1}^i x_{j2}^i \ldots x_{j(d-1)}^i\right]$ and $f$ is a function from $\mathbb{R}^{d-1} \rightarrow \mathbb{R}$. Every data point can be viewed as an input and output pair. Below we define the global knowledge which is the union of all the data of all the peers.

*Definition 3.1 (Global knowledge):* The **global knowledge** is the set of all inputs at time $t$ and is denoted by $\mathcal{G} = \bigcup_{i=1,\ldots,n} S_i$.

Henceforth we will drop the indices in $\mathcal{G}$.

In MR, the task is to learn the function $\widehat{f}(\overrightarrow{x})$ which "best" approximates $f(\overrightarrow{x})$ according to some measure such as least square. Now depending on the representation chosen for $\widehat{f}(\overrightarrow{x})$, various types of regression models (linear or nonlinear) can be developed. We leave this type specification as part of the problem statement for our algorithm, rather than an assumption.

For each data point $(\overrightarrow{x}, f(\overrightarrow{x}))$, the error between $\widehat{f}(\overrightarrow{x})$ and $f(\overrightarrow{x})$ can be computed as $\left[f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x})\right]^2$. Normally we require the error to be zero. However, since we have a dynamic data change scenario we relax this assumption and consider a solution to be admissible if the global error is less than $\epsilon$, where $\epsilon$ is a user chosen threshold. For peer $P_i$, this error $\mathcal{E}_i$ is a set of points in $\mathbb{R}$ i.e. $\mathcal{E}_i = \left\{\left[f(\overrightarrow{x_1^i}) - \widehat{f}(\overrightarrow{x_1^i})\right]^2, \left[f(\overrightarrow{x_2^i}) - \widehat{f}(\overrightarrow{x_2^i})\right]^2, \ldots\right\}$. The average error for $P_i$ is denoted by $\overline{\mathcal{E}_i} = \frac{1}{|\mathcal{E}_i|}\sum_j \left[f(\overrightarrow{x_j^i}) - \widehat{f}(\overrightarrow{x_j^i})\right]^2$.

Moreover, in our scenario, this error value is distributed across the peers — therefore a good estimate of the global error is the global average error *i.e.* $\mathcal{E}^{\mathcal{G}} = \frac{1}{n}\sum_i \overline{\mathcal{E}_i}$ over all the points in $\mathcal{G}$.

Peers communicate with one another by sending sets of points in $\mathbb{R}$ or statistics as defined in Section III-B. 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 $X_{j,i}$. Our next definition formally defines a message.

*Definition 3.2 (Message):* The **message** that peer $P_i$ needs to send to $P_j$ consists of a set of vectors and is denoted by $X_{i,j}$. Each vector is in $\mathbb{R}$ and the size of the set depends on the data that peer $P_i$ needs to send to $P_j$.

Below we show that for our case, sending the statistics of the set (such as mean and size) is sufficient. Now we define four entities which are crucial to our algorithm.

*Definition 3.3 (Knowledge):* The **knowledge** of $P_i$ is the union of $\mathcal{E}_i$ with $X_{j,i}$ for all $P_j \in \Gamma_i$ and is denoted by $\mathcal{K}_i = \mathcal{E}_i \cup \bigcup_{P_j \in \Gamma_i} X_{j,i}$.

*Definition 3.4 (Agreement):* 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 3.5 (Withheld knowledge):* The subtraction of the agreement from the knowledge is the **withheld knowledge** of $P_i$ with respect to a neighbor $P_j$ *i.e.* $\mathcal{W}_{i,j} = \mathcal{K}_i \setminus \mathcal{A}_{i,j}$.

We are interested in computing regression models defined on $\mathcal{G}$. Note that no peer has the global error or $\mathcal{E}^{\mathcal{G}}$. Therefore each peer will estimate $\mathcal{E}^{\mathcal{G}}$ based on only its local knowledge $\mathcal{K}_i$. These sets can be arbitrarily large. Hence in order to represent these sets efficiently, we define two statistics on each set: (1) the *average* which is the average of all the points in the respective sets (e.g. $\overline{\mathcal{K}_i}$, $\overline{\mathcal{A}_{i,j}}$, $\overline{\mathcal{W}_{i,j}}$, $\overline{X_{i,j}}$, $\overline{X_{j,i}}$ and $\overline{\mathcal{E}^{\mathcal{G}}}$), and (2) the *sizes* of the sets denoted by $|X_{i,j}|$, $|X_{j,i}|$, $|\mathcal{K}_i|$, $|\mathcal{A}_{i,j}|$, $|\mathcal{W}_{i,j}|$, and $\left|\mathcal{E}^{\mathcal{G}}\right|$. Instead of communicating the entire sets of points, each peer can

communicate only these two statistics for each set which is sufficient to represent them.

### B. Sufficient Statistics

Our algorithm relies on the fact that points sent by any peer $P_i$ to $P_j$ are never sent back to $P_i$. This can be done in several different ways such as message indexing, tagging and ensuring that the graph topology has no cycles. In this paper we take a simpler approach — we assume that a tree topology is imposed over the network. We could get around this assumption in one of two ways:

1) We can use a similar technique as proposed by Liss *et al.* [36] which extends the original majority voting algorithm for arbitrary network topology.
2) There exist several techniques in the literature for maintaining tree communication topology such as [37] (for wired networks) or [38] (for wireless networks).

If we assume that communication always takes place in an overlay tree topology, we can write the following expressions for the sizes of the sets:

1. $|\mathcal{A}_{i,j}| = |X_{i,j}| + |X_{j,i}|$
2. $|\mathcal{K}_i| = |\mathcal{E}_i| + \sum_{P_j \in \Gamma_i} |X_{j,i}|$, and
3. $|\mathcal{W}_{i,j}| = |\mathcal{K}_i| - |\mathcal{A}_{i,j}|$.

Similarly for the average of the sets we can write,

1. $\overline{\mathcal{A}_{i,j}} = \frac{|X_{i,j}|}{|\mathcal{A}_{i,j}|}\overline{X_{i,j}} + \frac{|X_{j,i}|}{|\mathcal{A}_{i,j}|}\overline{X_{j,i}}$
2. $\overline{\mathcal{K}_i} = \frac{|\mathcal{E}_i|}{|\mathcal{K}_i|}\overline{\mathcal{E}_i} + \sum_{P_j \in \Gamma_i} \frac{|X_{j,i}|}{|\mathcal{K}_i|}\overline{X_{j,i}}$
3. $\overline{\mathcal{W}_{i,j}} = \frac{|\mathcal{K}_i|}{|\mathcal{W}_{i,j}|}\overline{\mathcal{K}_i} - \frac{|\mathcal{A}_{i,j}|}{|\mathcal{W}_{i,j}|}\overline{\mathcal{A}_{i,j}}$

Note that, for any peer, any of these quantities can be computed based solely on its local data and what it gets from its immediate neighbors.

Next we formally state the problem definition.

### C. Problem Definition

*Problem 1.* [**MR Problem**] Given a time varying dataset $S_i$, a user-defined threshold $\epsilon$ and $\widehat{f}(\overrightarrow{x}) : \mathbb{R}^{d-1} \rightarrow \mathbb{R}$ to all the peers, the MR problem is to maintain a $\widehat{f}(\overrightarrow{x})$ at each peer such that, at any time $t$, $\mathcal{E}^{\mathcal{G}} < \epsilon$.

For ease of explanation, we decompose this task into two subtasks. First, given a representation of $\widehat{f}(\overrightarrow{x})$ to all the peers, we want to raise an alarm whenever $\mathcal{E}^{\mathcal{G}} > \epsilon$, where $\epsilon$ is a user-defined threshold. This is the *model monitoring problem*. Now if $\widehat{f}(\overrightarrow{x})$ no longer represents $f(\overrightarrow{x})$, we sample from the network (or even better do an in-network aggregation) to find an updated $\widehat{f}(\overrightarrow{x})$. This is the *model computation problem*. Mathematically, the subproblems can be formalized as follows.

*Problem 2.*[**Monitoring Problem**] Given $S_i$, and $\widehat{f}(\overrightarrow{x})$ to all the peers, the monitoring problem is to output 0 if $\mathcal{E}^{\mathcal{G}} < \epsilon$, and 1 otherwise, at any time $t$.

*Problem 3.*[**Computation Problem**] The model computation problem is to find a new $\widehat{f}(\overrightarrow{x})$ based on a sample of the data collected from the network.

Also note that the case for which the output is 0 can be defined as the region $C_\omega^- = \{x \in \mathbb{R} : 0 < x < \epsilon\}$. The region in which the output is 1 can be defined as $C_\omega^+ = \{x \in \mathbb{R} : \epsilon < x < \infty\}$ Further, let $C_\omega = \{C_\omega^+, C_\omega^-\}$. In order to ensure global correctness of the monitoring algorithm, we have transformed the thresholding problem to a geometric problem: we check if the global average error lies in $C_\omega^-$. In Section IV, we discuss a lemma relying on $C_\omega$ which will guarantee correctness of the monitoring algorithm.

### *D. Example*

In this section we illustrate the P2P MR algorithm. Let there be two peers $P_i$ and $P_j$. Let the regression model be linear in the regression coefficients: $a_0 + a_1x_1 + a_2x_2$, where $a_0, a_1$ and $a_2$ are the regression coefficients having values 1, 2 and -2 respectively and $x_1$ and $x_2$ are the two attributes of the data. The coefficients are given to all the peers. The data of peer $P_i$ is $S_i = \{(3, 1, 3.9), (0, -1, 3.6)\}$, where the third entry of each data point is the output generated according to the regression model. To this, we add 30% noise. Similarly, for peer $P_j$, $S_j = \{(1, 4, -6.5), (-3, 2, -9.1)\}$. Now for peer $P_i$, the squared error for each point is: $\mathcal{E}_i = \{(0.9)^2, (0.6)^2\}$. Similarly for $P_j$, the errors are $\mathcal{E}_j = \{(1.5)^2, (2.1)^2\}$. Hence $\overline{\mathcal{E}_i} = \left\{\frac{(0.9)^2+(0.6)^2}{2}\right\} = \{0.585\}$ and $\overline{\mathcal{E}_j} = \left\{\frac{(1.5)^2+(2.1)^2}{2}\right\} = \{3.33\}$. Assuming $\overline{X_{i,j}} = \overline{X_{j,i}} = 0$, for peer $P_i$, $\overline{\mathcal{K}_i} = \overline{\mathcal{E}_i} = \{0.585\}$. Similarly for peer $P_j$, $\overline{\mathcal{K}_j} = \overline{\mathcal{E}_j} = \{3.33\}$. Also the global

average error is $\overline{\mathcal{E}^{\mathcal{G}}} = \left\{\frac{(0.9)^2+(0.6)^2+(1.5)^2+(2.1)^2}{4}\right\} = \{1.9575\}$. In $\mathbb{R}$, the task is to determine if $1.9575 > \epsilon$ for a user defined $\epsilon$.

### *E. Local Algorithm*

Local algorithms, as defined by Das et al. [3], are parameterized by two quantities: (1) $\alpha$ – which is the number of neighbors a peer contacts in order to find answers to a query and (2) $\gamma$ – which is the total size of the response which a peer receives as the answer to all the queries executed throughout the lifetime of the algorithm. $\alpha$ can be a constant or a function parameterized by the size of the network while $\gamma$ can be parameterized by both the size of the data of a peer and the size of the network. Here we present the definition proposed by Das et al. [3].

*Definition 3.6 ($\alpha$-neighborhood of a vertex):* Let $G = (V, E)$ be the graph representing the network where $V$ denotes the set of nodes and $E$ represents the edges between the nodes. The **$\alpha$-neighborhood of a vertex** $v \in V$ is the collection of vertices at distance $\alpha$ or less from it in $G$: $\Gamma_v(\alpha, v, V) = \{u | dist(u, v) \leq \alpha\}$, where $dist(u, v)$ denotes the length of the shortest path in between $u$ and $v$ and the length of a path is defined as the number of edges in it.

*Definition 3.7 ($\alpha$-local query):* Let $G = (V, E)$ be a graph as defined in last definition. Let each node $v \in V$ store a data set $X_v$. An ***$\alpha$-local query*** by some vertex $v$ is a query whose response can be computed using some function $f(X_\alpha(v))$ where $X_\alpha(v) = \{X_v | v \in \Gamma_v(\alpha, v, V)\}$.

*Definition 3.8 ($(\alpha, \gamma)$-local algorithm):* An algorithm is called ***$(\alpha, \gamma)$-local*** if it never requires computation of a $\beta$-local query such that $\beta > \alpha$ and the total size of the response to all such $\alpha$-local queries sent out by a peer is bounded by $\gamma$. $\alpha$ can be a constant or a function parameterized by the size of the network while $\gamma$ can be parameterized by both the size of the data of a peer and the size of the network.

The idea is to design algorithms that offers bounded total communication cost per node and also spatially localized communication among the neighbors. We call such an ($\alpha$, $\gamma$)-local algorithm ***efficient*** if both $\alpha$ and $\gamma$ are either small constants or some slow growing functions (sublinear) with respect to its parameters. We prove that the regression monitoring algorithm is $(O(1), O(n))$-local in Section IV-B.

## IV. STEP 1: MONITORING REGRESSION MODEL

In MR monitoring problem, each peer is given a dataset $S_i$ and an estimate $\widehat{f}(\overrightarrow{x})$. Our goal is to monitor $\mathcal{E}^{\mathcal{G}}$.

We present here a local algorithm which monitors the regression coefficients by thresholding the average error. In our earlier work [33], we presented an algorithm for monitoring the L2 norm of the average vector distributed across a large number of peers. The algorithm outputs 0 if $\left|\left|\overrightarrow{\mathcal{G}}\right|\right| < \epsilon$ and 1 otherwise. The algorithm presented in [33] is prone to noise in the data since it communicates all the data for every data change. In this paper, we take care of that problem by applying a different condition for sending messages and ensuring that all data is not sent whenever a peer communicates. Rather, we keep some data (in the form of withheld knowledge) so that if the data changes later, the change is less noisy. Here we use a similar algorithm but in $\mathbb{R}$ and use a different set of conditions for sending messages in order to reduce the communication overhead in dynamically changing environments.

The regression monitoring algorithm guarantees eventual correctness, which means that once computation terminates, each peer computes the correct result as compared to a centralized setting. In a termination state, no messages traverse the network, and hence a peer can decide solely based on $\overline{\mathcal{K}_i}$, $\overline{\mathcal{A}_{i,j}}$, and $\overline{\mathcal{W}_{i,j}}$, if $\overline{\mathcal{E}^{\mathcal{G}}}$ is greater than or less than $\epsilon$. As stated by the Theorem below, if the following condition holds, the peer can stop sending messages and determine the correct output based solely on its local averages.

*Theorem 4.1:* **[Stopping Rule]** Let $P_1, \ldots, P_n$ be a set of peers connected to each other over a spanning tree $G(V, E)$. Let $\mathcal{E}^{\mathcal{G}}$, $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, and $\mathcal{W}_{i,j}$ be as defined in the previous section. Let $R$ be any region in $C_\omega$. 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 $\overline{\mathcal{E}^{\mathcal{G}}} \in R$.

*Proof:* [**Sketch]:**We omit the formal proof here. Simply speaking, the theorem can be proved by taking any two arbitrary peers and exchanging all of their withheld knowledge. We call this the unification step. After unifying all the peers it can be shown that $\overline{\mathcal{E}^{\mathcal{G}}} \in R$. Interested readers are referred to [39]. ■

The significance of Theorem 4.1 is that under the condition described $P_i$ can stop sending messages to its neighbors and output if $\overline{\mathcal{K}_i} < \epsilon$. The idea is to ensure that $\overline{\mathcal{K}_i}$ and $\overline{\mathcal{E}^{\mathcal{G}}}$ finally reside in the same region in $C_\omega$. If the result of the theorem holds for every peer, then Theorem

4.1 guarantees this is the correct solution; else, there must either be a message in transit, or some peer $P_k$ for whom the condition does not hold. Then either $P_k$ will send a message which will change its output or the message will be received, leading to a change in $\overline{\mathcal{K}_k}$ eventually. Thus eventual correctness is guaranteed. We formally prove this in Section IV-A.

**Input**: $\epsilon$, $C_\omega$, $S_i$, $\Gamma_i$ and $L$
**Output**: 0 if $\overline{\mathcal{K}_i} < \epsilon$, 1 otherwise
**Initialization**: Initialize vectors;
**if** MessageRecvdFrom $\left(P_j, \overline{X}, |X|\right)$ **then**
  $\overline{X_{j,i}} \leftarrow \overline{X}$;
  $|X_{j,i}| \leftarrow |X|$;
  Update vectors;
**end**
**if** $S_i$, $\Gamma_i$ *or* $\mathcal{K}_i$ *changes* **then**
  **forall** $P_j \in \Gamma_i$ **do**
    **if** $LastMsgSent > L$ *time units ago* **then**
      **if** $R = \emptyset$ **then**
        $\overline{X_{i,j}} \leftarrow \frac{|\mathcal{K}_i|\overline{\mathcal{K}_i} - |X_{j,i}|\overline{X_{j,i}}}{|\mathcal{K}_i| - |X_{j,i}|}$;
        $|X_{i,j}| \leftarrow |\mathcal{K}_i| - |X_{j,i}|$;
      **end**
      **if** $\overline{\mathcal{A}_{i,j}} \notin R$ *or* $\overline{\mathcal{W}_{i,j}} \notin R$ **then**
        Set $\overline{X_{i,j}}$ and $|X_{i,j}|$ such that $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}} \in R$;
      **end**
      **SendMessage**$\left(P_j, \overline{X_{i,j}}, |X_{i,j}|\right)$;
      $LastMsgSent \leftarrow L$;
      Update all vectors;
    **end**
    **else** Wait $L$ time units and then check again;
  **end**
**end**

**Algorithm 1**: Monitoring Regression Model.

Algorithm 1 presents the pseudo-code. The inputs to the algorithm are $S_i$, $\Gamma_i$, $\epsilon$ and $C_\omega$ and $L$. Each peer initializes its local statistics $\overline{\mathcal{K}_i}$, $\overline{\mathcal{A}_{i,j}}$ and $\overline{\mathcal{W}_{i,j}}$. A peer may need to send a message if its local data changes, if it receives a message or if the set of neighbors change. In any of these cases, the peer checks if the condition of the theorem holds. First peer $P_i$ finds the region $R \in C_\omega$ such that $\overline{\mathcal{K}_i} \in R$. 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; else it needs to set $\overline{X_{i,j}}$ and $|X_{i,j}|$ and send those, such that after the message is sent, the condition of the theorem holds for this peer. As we already pointed out that if a peer communicates all

of its data, then if 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, in which case we have to send everything. Note that other than these two cases, a peer need not send a message since the theorem guarantees eventual correctness. Similarly, whenever it receives a message ($\overline{X}$ and $|X|$), it sets $\overline{X_{j,i}} \leftarrow \overline{X}$ and $|X_{j,i}| \leftarrow |X|$. This may trigger another round of communication since its $\overline{\mathcal{K}_i}$ can now change.

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 $L$. Whenever a peer needs to send a message it checks if $L$ time units have passed since the last time it sent a message. If yes, it simply sends the message and notes the time. If not, it sets up a timer and initializes it to the time difference between $L$ and the time it had sent the last message. 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. This technique has been used elsewhere as well [33][17].

In the next two sections we discuss the correctness and locality of the multivariate regression monitoring algorithm.

### *A. Correctness*

In this section we prove that the regression monitoring algorithm is eventually correct. Theorem 4.2 formally proves the claim.

*Theorem 4.2:* **[Correctness]** The regression monitoring algorithm is **eventually correct**.

*Proof:* Each peer will continue to send messages and accumulate more and more of $\mathcal{E}^{\mathcal{G}}$ in each $\mathcal{K}_i$ until one of the two things happen: either for every peer, $\overline{\mathcal{K}_i} = \mathcal{E}^{\mathcal{G}}$ or for every $P_i$, both $\mathcal{K}_i$, $\mathcal{A}_{i,j}$, and $\mathcal{W}_{i,j}$ are in the same $R_\ell \in C_\omega$. In the former case, $\overline{\mathcal{K}_i} = \mathcal{E}^{\mathcal{G}}$, so every peer obviously computes the correct output. In the latter case, Theorem 4.1 dictates that $\mathcal{E}^{\mathcal{G}} \in R_\ell$. Since the function output (in this case 0 or 1) does not change inside each of these regions in $C_\omega$, and $\mathcal{E}^{\mathcal{G}}$ and $\overline{\mathcal{K}_i}$ lie inside the same region, the output of the test $\mathcal{E}^{\mathcal{G}} < \epsilon$ will be the same as $\overline{\mathcal{K}_i} < \epsilon$. Therefore in either of the cases, the regression monitoring algorithm is correct. ■

### B. Locality

In this section we claim that the regression monitoring algorithm is $(O(1), O(n))$-local. The art of measuring $(\alpha, \gamma)$-locality of algorithms is at its infancy. An attempt has been made to define locality with respect to the *Veracity Radius* of an aggregation problem [35]. However this method does not extend well to algorithms that contain randomness (e.g., in message scheduling) or to dynamic data and topology. Considering the $(\alpha, \gamma)$ framework we defined earlier, there always exist problem instances for which any eventually correct algorithm (e.g. [16][31][33][40][41] and the one described in this paper) will have worst case $\gamma = O(n)$ (as shown in Theorem 4.4), where $n$ is the size of the network. While $O(n)$ is the upper bound on the communication complexity, more accurate bounds on $\gamma$ can be developed by identifying the specific problems and input instances. We feel that there is an intrinsic relation between $\gamma$ and $\epsilon$. For example increasing $\epsilon$ decreases $\gamma$ though it needs to be investigated further.

*Lemma 4.3:* Considering a two node network, $P_i$ and $P_j$, the maximum number of messages exchanged between them to come to a consensus about the correct output is 2.

*Proof:* Using the notations defined earlier, let $\overline{\mathcal{K}_i} \in R_k$, $\overline{\mathcal{K}_j} \in R_\ell$ and $\mathcal{E}^{\mathcal{G}} \in R_m$, where $R_m, R_k, R_\ell \in C_\omega$ and $k \neq \ell$ and $m = k$ or $\ell$. Considering an initialization state, where $\overline{X_{i,j}} = \overline{X_{j,i}} = 0$ such that $\overline{\mathcal{A}_{i,j}} = 0 = \overline{\mathcal{A}_{j,i}}$. In this case the condition of Theorem 4.1 does not hold for either $P_i$ or $P_j$. Without loss of generality let us assume that the conditions are violated at $P_i$. It will send all of its data *i.e.* $\overline{\mathcal{K}_i}$ to $P_j$ which will enable $P_j$ to correctly compute $\mathcal{E}^{\mathcal{G}}$ (since $\mathcal{E}^{\mathcal{G}}$ is a convex combination of $\overline{\mathcal{K}_i}$ and $\overline{\mathcal{K}_j}$). On receiving $\overline{\mathcal{K}_i}$ from $P_i$, $P_j$ will apply the conditions of Theorem 4.1. Since clearly $\overline{\mathcal{K}_j} = \mathcal{E}^{\mathcal{G}} \in R_m$ but $\overline{\mathcal{A}}_{j,i} = \overline{\mathcal{K}_i} \in R_k$, the condition of the theorem dictates it to send a message to $P_i$ and it will send all the data which it has not received from $P_i$ i.e. $\overline{\mathcal{K}_j}$. At this point both $P_i$ and $P_j$ have both $\overline{\mathcal{K}_i}$ and $\overline{\mathcal{K}_j}$. Hence they can compute $\mathcal{E}^{\mathcal{G}}$ correctly. Therefore the number of messages exchanged is 2. ■

Our next theorem bounds the total number of messages sent by the regression monitoring algorithm. Because of the dependence on the data, counting the number of messages in a data independent manner for such an asynchronous algorithm seems extremely difficult. Therefore in the following theorem (Theorem 4.4), we find the upper bound of the number of messages exchanged by any peer when the data of all the peer changes.

*Lemma 4.4:* **[Communication Complexity]** Let $D_t$ be a state of the network at time $t$ where for every $P_i$, $\overline{\mathcal{K}_i} \in R_\ell$, $R_\ell \in C_\omega$. Hence $\mathcal{E}^{\mathcal{G}} \in R_\ell$ as well and thus the peers have converged to the correct result. Let at time $t' > t$ the data of each peer changes. Without loss of generality, let us assume that at time $t'$, $\overline{\mathcal{K}_i} \in R_i$ where each $R_i \in C_\omega$. Let us also assume that $\mathcal{E}^{\mathcal{G}} \in R_g$, where $g \notin \{1 \ldots n\}$. The maximum number of messages sent by any peer $P_i$ is $(n-1) \times (|\Gamma_i| - 1)$ in order to ensure $\overline{\mathcal{K}_i} \in R_g$.

*Proof:* It is clear that the output of each peer will be correct only when each $\overline{\mathcal{K}_i} = \mathcal{E}^{\mathcal{G}}$. This will only happen when each $P_i$ has communicated with all the peers in the network i.e. $\overline{\mathcal{K}_i} = \sum_{i=1}^{n} \overline{\mathcal{K}_i}$. Since the regression monitoring algorithm only communicates with immediate neighbors, in the worst case any peer $P_i$ will be updated with each value of $\overline{\mathcal{K}_j}$, $j \neq i$ one at a time. Every time $P_i$ gets one $\overline{\mathcal{K}_j}$, it communicates with all its neighbors except the one from which it got $\overline{\mathcal{K}_j}$. This process can be repeated in the worst case for $(n-1)$ times in order to get all the $\overline{\mathcal{K}_j}$'s. At every such update, $P_i$ will communicate with $|\Gamma_i| - 1$ neighbors. Therefore, the total number of messages sent by $P_i$ is $(n-1) \times (|\Gamma_i| - 1)$. ■

Our next theorem shows that the multivariate regression monitoring algorithm is $(O(1), O(n))$-local.

*Theorem 4.5:* **[Locality]** The multivariate regression monitoring algorithm is $(O(1), O(n))$-local.

*Proof:* The multivariate regression algorithm is designed to work by communicating with immediate neighbors of a peer only. Hence by design, $\alpha = 1$.

From Lemma 4.4, we know that $\gamma = O(n)$. Hence, the multivariate regression monitoring algorithm is $(O(1), O(n))$-local. ■

Although the worst case communication complexity $\gamma$ is $O(n)$, for many interesting problem instances $\gamma$ is a small constant and independent of the size of the network as corroborated by our extensive experimental results.

### *C. An Alternate Approach*

In the previous section we used L2-norm monitoring as the building block for monitoring the regression coefficients. In this section we discuss another primitive *viz.* majority voting protocol which can be used for the same matter.

Majority voting protocol for large P2P systems was proposed by Wolff and Schuster [31]. In its basic form, each peer $P_i$ contains a real number $\delta^i$ and the objective is to determine whether $\Delta = \sum_{i \in V} \delta^i \geq \epsilon'$, where $\epsilon'$ is a user chosen threshold.

The task of computing if $\Delta > \epsilon'$ can be achieved by the following algorithm. For peers $P_i$ and $P_j$, let $\delta^{i,j}$ denote the most recent message (a real number) peer $P_i$ sends to $P_j$. Similarly $\delta^{j,i}$ denotes the last message received by $P_i$ from $P_j$. Now using a similar mnemonic as done in the previous sections, the knowledge of $P_i$ is $\Delta^i = \delta^i + \sum_{P_j \in \Gamma_i} \delta^{j,i}$. Similarly, the agreement of peer $P_i$ and $P_j$ is $\Delta^{i,j} = \delta^{i,j} + \delta^{j,i}$, for each neighbor $P_j \in \Gamma_i$. The algorithm is entirely event based — an event at $P_i$ can be one of the following: (i) $P_i$ is initialized (enters the network or otherwise begins computation of the algorithm); (ii) $P_i$ experiences a data change $\delta^i$ or a change of its neighborhood, $\Gamma_i$; (iii) $P_i$ receives a message from a neighbor $P_j$. If any of these events occur, peer $P_i$ needs to check conditions on its knowledge and agreement to determine if a message needs to be sent to $P_j$. It can be shown that peer $P_i$ needs to send a message to $P_j$ only if the following test returns true: $(\Delta^{i,j} \geq 0 \wedge \Delta^{i,j} > \Delta^i) \vee (\Delta^{i,j} < 0 \wedge \Delta^{i,j} < \Delta^i)$. Since all these events are local to a peer, the algorithm requires no form of global synchronization and hence can be used for our regression monitoring algorithm.

What is left to discuss is what $P_i$ sends to $P_j$, if the conditions dictate so. $P_i$ first sets $\delta^{i,j}$ to $\beta\Delta^i - \delta^{j,i}$ (thereby making $\Delta^{i,j} = \beta\Delta^i$) and sends it to $j$, where $\beta$ is a fixed parameter between 0 and 1. Reducing $\beta$ reduces the number of messages in a dynamic setup while increasing the convergence time. This mechanism replicates the one used by Wolff *et al.* in [33] and Bhaduri *et al.* [17]. The pseudo-code is presented in Algorithm 2.

In order to use this protocol for regression monitoring, the following steps need to be taken:

- The input $\delta^i$ for each peer should be the average error calculated on its own local dataset $S_i$ *i.e.* $\delta^i = \overline{\mathcal{E}_i} = \frac{1}{|S_i|} \sum_{\overrightarrow{x} \in S_i} \left[ f(\overrightarrow{x}) - \widehat{f}(\overrightarrow{x}) \right]^2$.
- Choose $\epsilon' = \epsilon \times n$, where $n$ is the total number of nodes in the network.

Other than these two changes, the majority voting algorithm can be used for regression monitoring without any further change.

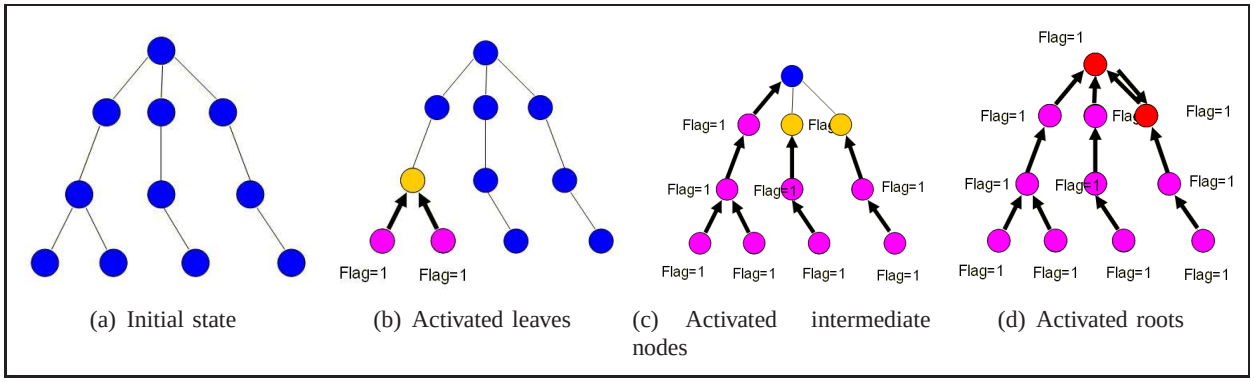
In the next section we discuss the algorithm for computing the regression coefficients.

## V. Step 2: Computing Regression Model

The regression monitoring algorithm presented in the earlier section can be viewed as a flag which is raised by a peer whenever $\mathcal{E}^{\mathcal{G}} > \epsilon$. In this section we discuss how the peers collaborate to find a new $\widehat{f}(\overrightarrow{x})$ using a convergecast-broadcast technique.

The basic idea is to use the *convergecast* phase to sample data from the network to a central post and compute, based on this sample, a new $\widehat{f}(\overrightarrow{x})$. The *broadcast* phase distributes this $\widehat{f}(\overrightarrow{x})$ to the network. The monitoring algorithm now monitors the quality of the result. The efficiency and correctness of the monitoring algorithm allows a very simple sampling technique to be used – if an ill-fit model is built at all, it will soon be detected by the local algorithm resulting in another round of convergecast in the worst case. Another point to note is that, in our convergecast-broadcast process, we do not specify the root of the convergecast tree. Rather we let the network structure (edge delays and data skewness) decide it. This is significant since it ensures (1) decentralized control, (2) load balancing, and (3) robustness against a single point of failure.

In the convergecast phase there are two main parameters. Each peer maintains a user selected alert mitigation constant, $\tau$ and the sample size. An alert should be stable for a given period of time $\tau$ before the peer can send its data, thereby preventing a possibly false alarm from propagating. In order to do this, the peer relies on the underlying monitoring algorithm. If the monitoring algorithm raises a flag, the peer notes the time, and sets a timer to $\tau$ time units. If the timer expires, or a data message is received from one of its neighbors, $P_i$ first checks if there is an existing alert and if it has been recorded $\tau$ or more time units ago. If so, it counts the number of neighbors from which it has received data messages. Once it receives data messages from all of its neighbors, the peer computes a new regression model $\widehat{f}(\overrightarrow{x})$ based on the sample it received and sends it to itself. It then moves to the broadcast phase and sends $\widehat{f}(\overrightarrow{x})$ to all its neighbors. On the other hand, if it has received data messages from all but one of the neighbors then it takes a uniform sample (of user-defined size) from its own data and the data it has received from its neighbors. It then forwards the sample to the peer from which it has not received data and marks its state as broadcast. The peer does nothing if it has not received data from two or more neighbors. Note that, at each peer, the sampling technique is such that, each data point gets an equal chance of being included in the sampled data set. We do this by properly weighing every




(a) Initial state (b) Activated leaves (c) Activated intermediate nodes (d) Activated roots

Fig. 1. Convergecast and broadcast through the different steps. In subfigure 1(a), the peers do not raise a flag. In subfigure 1(b), the two leaves raise their flags and send their data up (to the parent) as shown using arrows. Figure 1(c) shows an intermediate step. Finally, the roots (two of them) become activated in subfigure 1(d) by exchanging data with each other.

data point by size of the subtree from which the sample was received.

The broadcast phase is fairly straightforward. Every peer which receives the new $\widehat{f}(\overrightarrow{x})$, restarts a new regression monitoring algorithm with this new $\widehat{f}(\overrightarrow{x})$. It then, sends the new $\widehat{f}(\overrightarrow{x})$ to its other neighbors and changes the status to convergecast. There could be one situation in which a peer receives a new $\widehat{f}(\overrightarrow{x})$ when it is already in the broadcast phase. This is when two neighbor peers concurrently become roots for the convergecast tree. To break this tie, we select the $\widehat{f}(\overrightarrow{x})$ to propagate the root of which has a higher id. Figure 1 shows a snap-shot of the convergecast broadcast steps as it progresses up the communication tree. The pseudo-code is presented in Algorithm 3.

## VI. Special case : Linear Regression

In many cases, sampling from the network is communication intensive. We can find the coefficients using an in-network aggregation if we choose to monitor a widely used regression model *viz.* linear regression (linear with respect to the parameters or the unknown weights).

Let the global dataset over all the peers be denoted by:

$$\mathcal{G} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1(d-1)} & f(\overrightarrow{x_1}) \\ x_{21} & x_{22} & \dots & x_{2(d-1)} & f(\overrightarrow{x_2}) \\ \vdots & \vdots & & \vdots & \vdots \\ x_{j1} & x_{j2} & \dots & x_{j(d-1)} & f(\overrightarrow{x_j}) \\ \vdots & \vdots & & \vdots & \vdots \\ x_{|\mathcal{G}|1} & x_{|\mathcal{G}|2} & \dots & x_{|\mathcal{G}|(d-1)} & f(\overrightarrow{x_{|\mathcal{G}|}}) \end{pmatrix}$$

where $\overrightarrow{x_j} = \left\{x_{j1}x_{j2}\dots x_{j(d-1)}\right\}$.

In MR, the idea is to learn a function $\widehat{f}(\overrightarrow{x_j})$ which approximates $f(\overrightarrow{x_j})$ for all the data points in $\mathcal{G}$. For linear regression, that function $\widehat{f}(\overrightarrow{x_j})$ is chosen to be a linear function i.e. a $d-1$ degree polynomial fitted to the input attribute points $\left\{x_{j1}x_{j2}\dots x_{j(d-1)}\right\}$ $\forall j = 1$ to $|\mathcal{G}|$. More specifically, the linear model which we want to fit be: $\widehat{f}(\overrightarrow{x_j}) = a_0 + a_1x_{j1} + a_2x_{j2} + ... + a_{j(d-1)}x_{d-1}$, where $a_i$'s are the coefficients that need to be estimated from the global dataset $\mathcal{G}$. We drop the cross terms involving $x_{jk}$ and $x_{j\ell}$ for simplicity $\forall k, \ell \in [1..(d-1)]$.

For every data point in the set $\mathcal{G}$, the squared error is:

$$\begin{aligned} E_1 &= \left[f\left(\overrightarrow{x_1}\right) - a_0 - a_1x_{11} - a_2x_{12} - \dots - a_{d-1}x_{1(d-1)}\right]^2 \\ E_2 &= \left[f\left(\overrightarrow{x_1}\right) - a_0 - a_1x_{21} - a_2x_{22} - \dots - a_{d-1}x_{2(d-1)}\right]^2 \\ &\vdots \\ E_{|\mathcal{G}|} &= \left[f\left(\overrightarrow{x_{|\mathcal{G}|}}\right) - a_0 - a_1x_{|\mathcal{G}|1} - a_2x_{|\mathcal{G}|2} - \dots - a_{d-1}x_{|\mathcal{G}|(d-1)}\right]^2 \end{aligned}$$

Thus the total square error over all the data points is

$$SSE = \sum_{j=1}^{|\mathcal{G}|} E_j = \sum_{j=1}^{|\mathcal{G}|} \left[f\left(\overrightarrow{x_j}\right) - a_0 - a_1x_{j1} - a_2x_{j2} - \dots - a_{d-1}x_{j(d-1)}\right]^2$$

For linear regression, closed form expressions exist for finding the coefficients $a_i$'s by finding the partial derivatives of SSE with respect to the $a_i$'s and setting them to zero:

$$\begin{aligned}
\frac{\partial}{\partial a_0}SSE &= 2\sum_{j=1}^{|\mathcal{G}|}\left[f\left(\overrightarrow{x_j}\right) - a_0 - a_1x_{j1} - a_2x_{j2} - \ldots - a_{d-1}x_{j(d-1)}\right](-1) = 0 \\
\frac{\partial}{\partial a_1}SSE &= 2\sum_{j=1}^{|\mathcal{G}|}\left[f\left(\overrightarrow{x_j}\right) - a_0 - a_1x_{j1} - a_2x_{j2} - \ldots - a_{d-1}x_{j(d-1)}\right](-x_{j1}) = 0 \\
\frac{\partial}{\partial a_2}SSE &= 2\sum_{j=1}^{|\mathcal{G}|}\left[f\left(\overrightarrow{x_j}\right) - a_0 - a_1x_{j1} - a_2x_{j2} - \ldots - a_{d-1}x_{j(d-1)}\right](-x_{j2}) = 0 \\
&\vdots \\
\frac{\partial}{\partial a_{d-1}}SSE &= 2\sum_{j=1}^{|\mathcal{G}|}\left[f\left(\overrightarrow{x_j}\right) - a_0 - a_1x_{j1} - a_2x_{j2} - \ldots - a_{d-1}x_{j(d-1)}\right](-x_{j(d-1)}) = 0
\end{aligned}$$

In the matrix form this can be written as:

$$\begin{pmatrix}
|\mathcal{G}| & \sum_{j=1}^{|\mathcal{G}|} x_{j1} & \ldots & \sum_{j=1}^{|\mathcal{G}|} x_{j(d-1)} \\
\sum_{j=1}^{|\mathcal{G}|} x_{j1} & \sum_{j=1}^{|\mathcal{G}|} (x_{j1})^2 & \ldots & \sum_{j=1}^{|\mathcal{G}|} x_{j1} * x_{j(d-1)} \\
\vdots & \vdots & \ddots & \vdots \\
\sum_{j=1}^{|\mathcal{G}|} x_{j(d-1)} & \sum_{j=1}^{|\mathcal{G}|} x_{j(d-1)} * x_{j1} & \ldots & \sum_{j=1}^{|\mathcal{G}|} (x_{j(d-1)})^2
\end{pmatrix}$$

$$\times \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{|\mathcal{G}|} f(\overrightarrow{x_j}) \\ \sum_{j=1}^{|\mathcal{G}|} f(\overrightarrow{x}_j) x_{j1} \\ \vdots \\ \sum_{j=1}^{|\mathcal{G}|} f(\overrightarrow{x}_j) x_{j(d-1)} \end{pmatrix} \Rightarrow \mathbf{Xa} = \mathbf{Y}$$

Therefore for computing the matrix (or more appropriately vector) **a**, we need to evaluate the matrices **X** and **Y**. This can be done in a communication efficient manner by noticing that the entries of these matrices are simply sums. Consider the distributed scenario where $\mathcal{G}$ is distributed among $n$ peers $S_1, S_2, \ldots, S_n$. Any entry of **X**, say $\sum_{j=1}^{|\mathcal{G}|}(x_{j1})^2$, can be decomposed as

$$\sum_{j=1}^{|\mathcal{G}|}(x_{j1})^2 = \underbrace{\sum_{x_{j1}\in S_1}(x_{j1})^2}_{\text{for } S_1} + \underbrace{\sum_{x_{j1}\in S_2}(x_{j1})^2}_{\text{for } S_2} + \ldots + \underbrace{\sum_{x_{j1}\in S_n}(x_{j1})^2}_{\text{for } S_n}$$

Therefore each entry of $\mathbf{X}$ and $\mathbf{Y}$ can be computed by simple sum over all the peers. Thus, instead of sending the raw data in the convergecast round, peer $P_i$ can forward a locally computed matrix $\mathbf{X}_i$ and $\mathbf{Y}_i$. Peer $P_j$, on receiving this, can forward a new matrix $\mathbf{X}_j$ and $\mathbf{Y}_j$ by aggregating, in a component-wise fashion, its local matrix and the received ones. Note that the avoidance of the sampling technique ensures that the result is exactly the same compared to a centralized setting.

**Communication Complexity:** Next we prove a lemma which states the communication complexity of computing the linear regression model.

*Lemma 6.1:* The communication complexity of computing a linear regression model is only dependent on the degree of the polynomial ($d$) and is independent of the number of data points i.e. $|\mathcal{G}|$.

*Proof:* As shown in Section VI, the task of computing the regression coefficients $\{a_0, a_1, \ldots a_{d-1}\}$ can be reduced to computing the matrices $\mathbf{X}_i$ and $\mathbf{Y}_i$. The dimensionality of $\mathbf{X}_i$ $d.d = d^2$. Similarly the dimensionality of $\mathbf{Y}_i$ $d.1 = d$. Therefore the total communication complexity is $O(d^2)$, which is independent of the size of the dataset $|\mathcal{G}|$. ■

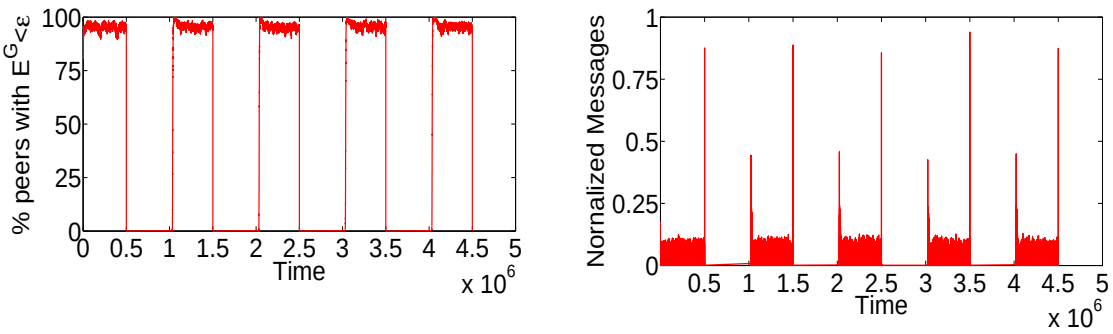
The efficiency of the convergecast process is due to the fact that $d \ll |\mathcal{G}|$. Hence there can be significant savings in terms of communication by not communicating the raw data.

## VII. Experimental Results

In this section we discuss the experimental setup and analyze the performance of the P2P regression algorithm.

### A. Experimental Setup

We have implemented our algorithms in the Distributed Data Mining Toolkit (DDMT) [42] developed by the DIADIC research lab at UMBC. We use topological information generated by the *Barabasi Albert (BA)* model in BRITE [43] since it is often considered a reasonable




(a) Percentage of peers with $\overline{\mathcal{K}_i} < \epsilon$ (b) Monitoring Cost

Fig. 2. A typical experiment is run for 10 equal length epochs. Quality and overall cost are measured across the entire experiment – including transitional phases. The monitoring cost is measured on the last $80\%$ of every epoch, in order to ignore transitional effects.

model for the internet. BA also defines delay for network edges, which is the basis for our time measurement[1]. On top of the network generated by BRITE, we overlay a communication tree.

### *B. Data Generation*

The input data of a peer is a vector $(x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$, where the first $d-1$ dimensions correspond to the input variables and the last dimension corresponds to the output. We have conducted experiments on both linear and non-linear regression models. For the linear model, the output is generated according to $x_d = a_0 + a_1x_1 + a_2x_2 + \ldots + a_{d-1}x_{d-1}$. We have used three functions for the non-linear model: (1) $x_3 = a_0 + a_1a_2x_1 + a_0a_1x_2$ (multiplicative), (2) $x_3 = a_0 \times sin(a_1 + a_2x_1) + a_1 \times sin(a_2 + a_0x_2)$ (sinusoidal) and (3) $x_3 = a_0\sqrt{x_0a_1} + a_1\sqrt{x_1a_0}$ (square root). Every time a simulated peer needs an additional data point, it chooses the values of $x_1, x_2, \ldots x_{d-1}$, each independently in the range -100 to +100. Then it generates the value of the target variable $x_d$ using any of the above functions and adds a uniform noise in the range 5 to 20% of the value of the target output. The regression weights $a_0, a_1, \ldots, a_{d-1}$'s are changed randomly at controlled intervals to create an epoch change.

### *C. Measurement Metric*

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

[1]Wall time is meaningless when simulating thousands of computers on a single PC.

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 $\epsilon$ and hence lies in between those two. Moreover, for each quality graph in Figures 3, 4, 5, 6, 7 and 8 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.

### *D. Typical Experiments*

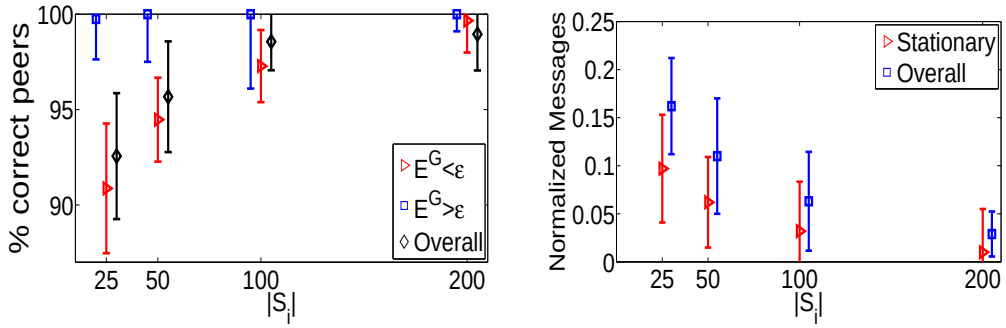
A typical experiment is shown in Figure 2. In all the experiments, about 4% of the data of each peer is changed every 1000 simulator ticks. Moreover, after every $5 \times 10^5$ simulator ticks, the data distribution is changed. Therefore there are two levels of data change — (1) every 1000 simulator ticks we sample 4% of new data from the same distribution (stationary change) and (2) every $5 \times 10^5$ 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 2(a) shows that for the first epoch, the quality is very high (nearly 96%). After $5 \times 10^5$ 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 2(b) shows that the monitoring cost is low throughout the experiment if we ignore the transitional effects.

### *E. Results: Regression Monitoring*

There are four external parameters which can influence the behavior of the regression monitoring algorithm: size of local buffer $|S_i|$, the threshold $\epsilon$, size of the leaky bucket $L$ and noise in the data. Apart from these there are also the system size (number of peers) and dimensionality of the multivariate regression problem which can affect performance. In this section we present the quality (less than *i.e.* $\mathcal{E}^{\mathcal{G}} < \epsilon$, greater than *i.e.* $\mathcal{E}^{\mathcal{G}} > \epsilon$ and overall) and cost of the algorithm w.r.t. different parameters. Note that, unless otherwise stated, we have used the following default values for the different parameters: number of peers = 1000, $|S_i| = 50$, $\epsilon = 1.5$, $d = 10$, $k = 8$ and $L = 500$ (where the average edge delay is about 1100 time units). As we have already stated, independent of the regression function chosen, the underlying monitoring problem is always in $\mathbb{R}$. The results reported in this section are with respect to linear model since it is the most widely used regression model. Results of monitoring more complex models are reported in the next section.

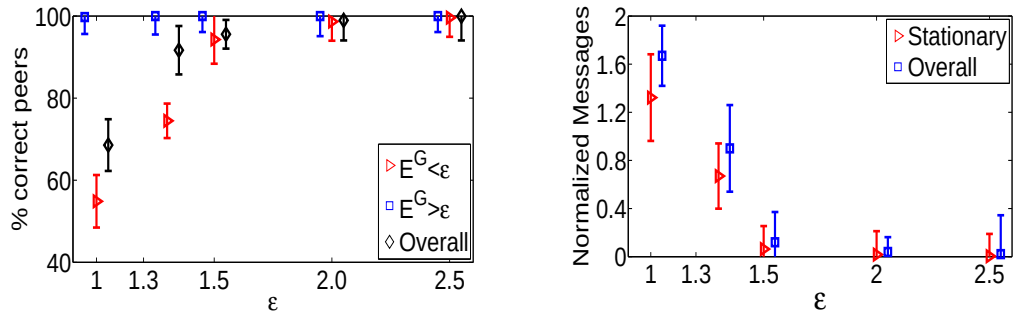
Figures 3(a) and 3(b) show the quality and cost of the algorithm as the size of local buffer is changed. As expected, the quality when the average is less than $\epsilon$ increases and the cost decreases as the size of buffer increases. The other quality is very high throughout. This stems from the fact that, with the noise in the data, it is easy for a peer to get flipped over when it is checking for less than $\epsilon$. On the other hand, noise cannot change the belief of the peer when the average is greater than $\epsilon$. In the second set of experiments, we varied $\epsilon$ from 1.0 to 2.5 (Figure 4(a) and 4(b)). Here also, the quality increases as $\epsilon$ is increased. This is because with increasing $\epsilon$, there is a bigger region in which to bound the global average. This is also reflected with decreasing number of messages. Note that, even for $\epsilon = 1.0$, the normalized messages are around 1.6, which is far less than the theoretical maximum of 2 (assuming two neighbors per peer). The third set of experiments analyzes the effect of leaky bucket $L$. As shown in Figure 5(a) quality does not depend on $L$, while Figure 5(b) shows that the cost decreases slowly with increasing $L$. Figures 6(a) and 6(b) depict the dependence of the noise on the monitoring algorithm. Quality degrades and cost increases with increasing noise. This is expected, since with increasing noise a peer is more prone to random effects. This effect can, however, be nullified by using a large buffer or bigger $\epsilon$.

Our next experiment analyzes the scalability of the monitoring algorithm w.r.t the number



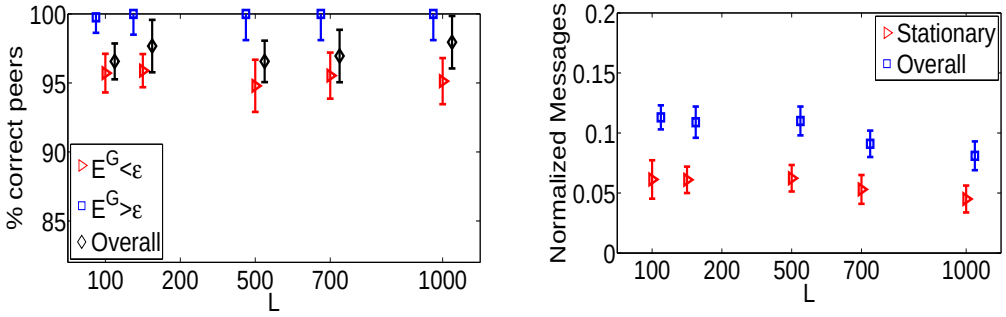

(a) Quality vs. $|S_i|$ (b) Cost vs. $|S_i|$

Fig. 3. Behavior of the monitoring algorithm with changes in the size of the dataset.



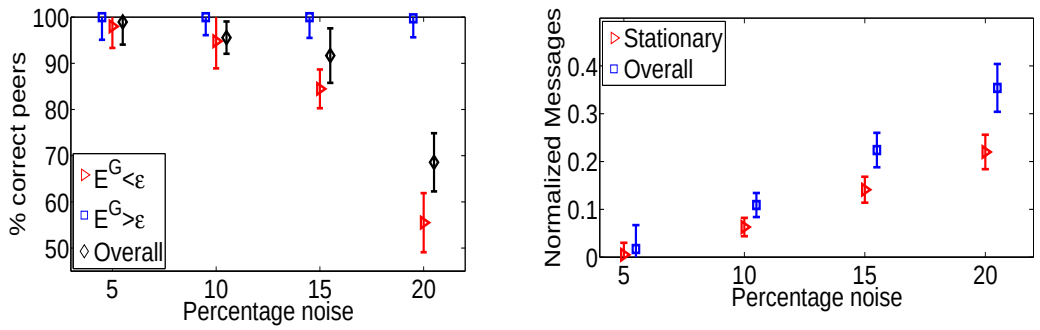

(a) Quality vs. $\epsilon$ (b) Cost vs. $\epsilon$

Fig. 4. Behavior of the monitoring algorithm with changes in $\epsilon$.



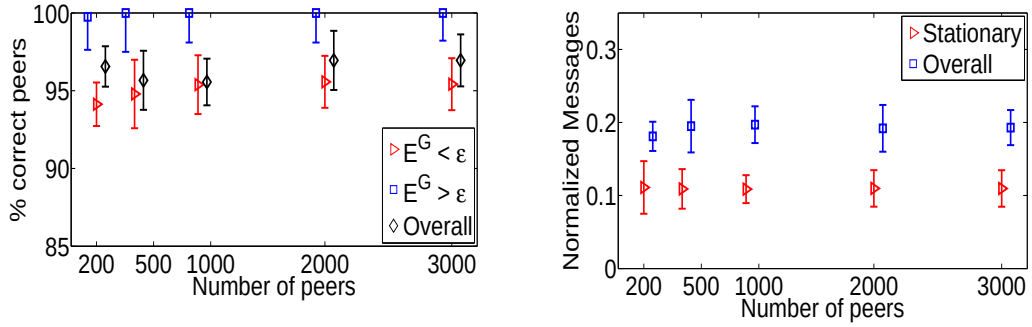

(a) Quality vs. $L$ (b) Cost vs. $L$

Fig. 5. Behavior of the monitoring algorithm with size of $L$.




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

Fig. 6. Behavior of the monitoring algorithm with variation of noise in the data.

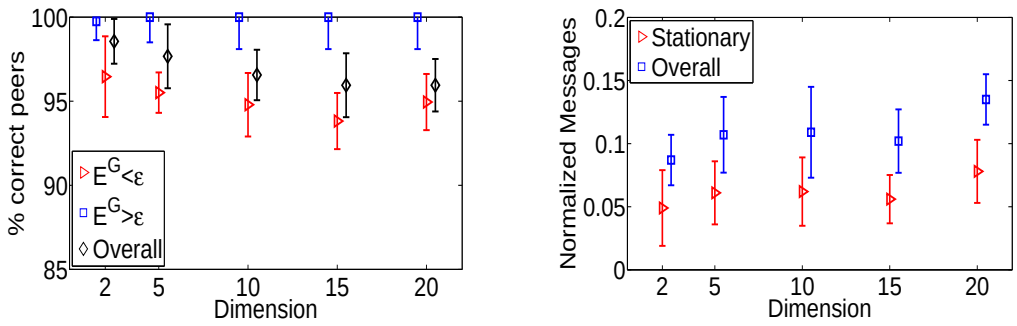



(a) Quality vs. number of peers (b) Cost vs. number of peers

Fig. 7. Scalability with respect to number of peers.

of peers and dimension of the multivariate problem. As Figures 7(a) and 7(b) show, both the quality and cost of the algorithm converge to a constant as the number of peers increase. This is a typical behavior of local algorithms. For any peer, since the computation is dependent on the result from only a handful of its neighbors, the overall size of the network does not degrade the quality or cost. Similarly, Figures 8(a) and 8(b) show that the quality or the cost does not depend on the dimension of the multivariate problem either. This independence of the quality and cost can be explained by noting that the underlying monitoring problem is in $\mathbb{R}$. Therefore for a given problem, the system size or dimensionality of the problem has no effect on the quality or the cost.

Overall, the results show that the monitoring algorithm offers extremely good quality, incurs low monitoring cost and has high scalability.




(a) Quality vs. dimension

(b) Cost vs. dimension

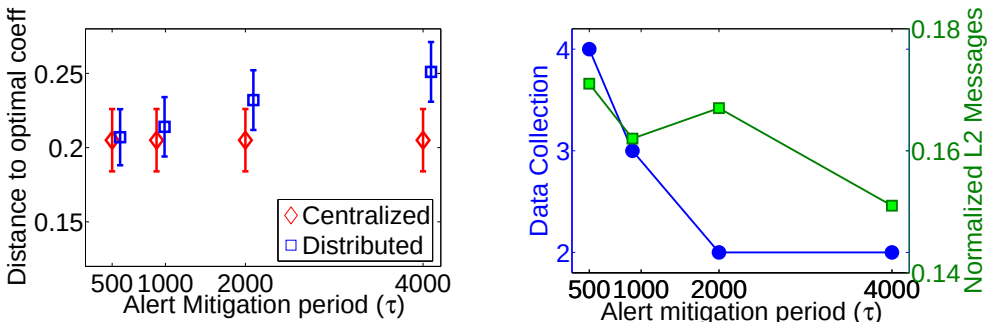
Fig. 8. Scalability with respect to dimension of the multivariate problem.

### *F. Results: Regression Models*

Our next set of experiments measure the quality of the regression model computed by our algorithm against a centralized algorithm having access to the entire data. There are two important parameters to be considered here – (1) the alert mitigation constant ($\tau$) and (2) the sample size (for non-linear regression). For computing the non-linear regression coefficients, we have implemented the Nelder-Mead simplex method [44].

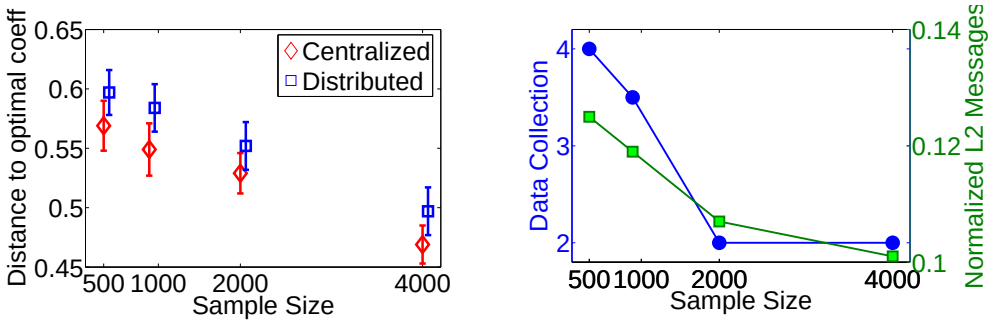
We have conducted experiments on three datasets. Figures 9(a), 10(a) and 11(a) presents two sets of error bars. The square markers show the L2 norm distance between the distributed coefficients and the actual ones. Also shown in each figure is the L2 norm distance between the coefficients found by a centralized algorithm and the actual ones (diamond markers). The first pair of figures, Figures 9(a) and 9(b) show the results of computing a linear regression model. Our aim is to measure the effect of variation of alert mitigation period $\tau$ on quality and cost. As shown in Figure 9(a), the quality of our algorithm deteriorates as $\tau$ increases. This is because, on increasing $\tau$, a peer builds a model later and therefore is inaccurate for a longer intermediate period. Figure 9(b) shows that the number of data collection rounds (dot markers) decrease from four times to twice per epoch. This results from a decrease in the number of false alerts. Also shown are monitoring messages (green squares).

Figures 10(a) and 10(b) analyzes the quality of our algorithm while computing a non-linear multiplicative regression model *viz.* $x_3 = a_0 + a_1 a_2 x_1 + a_0 a_1 x_2$. Figure 10(a) presents the quality as other parameter *viz.* sampling size is varied. As expected, the results from the distributed




(a) Quality for linear model (b) Cost for linear model

Fig. 9. Quality and cost of computing regression coefficients for a linear model.




(a) Quality for multiplicative model (b) Cost for multiplicative model

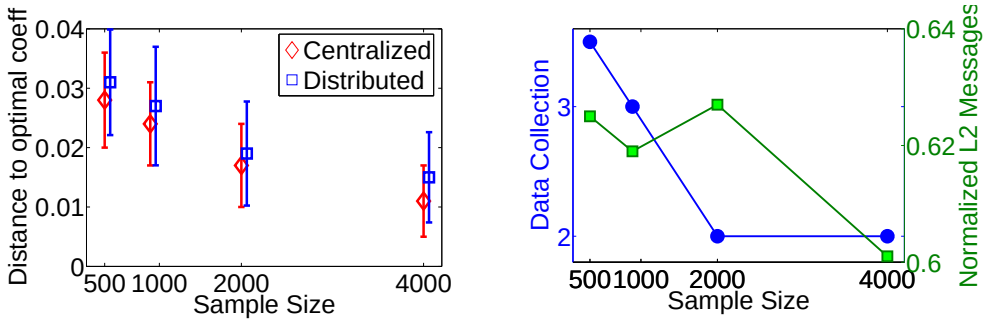
Fig. 10. Quality and cost of computing regression coefficients for $x_3 = a_0 + a_1 a_2 x_1 + a_0 a_1 x_2$.

and centralized computations converge with increasing sample size. Also the number of data collection rounds as depicted in Figure 10(b) decrease as sample size is increased.

The third pair of figures, Figures 11(a) and 11(b) show the same results for a sinusoidal model: $x_3 = a_0 * sin(a_1 + a_2 x_1) + a_1 * sin(a_2 + a_0 x_2)$. Here also the quality becomes better and the cost decreases as the sample size is increased.

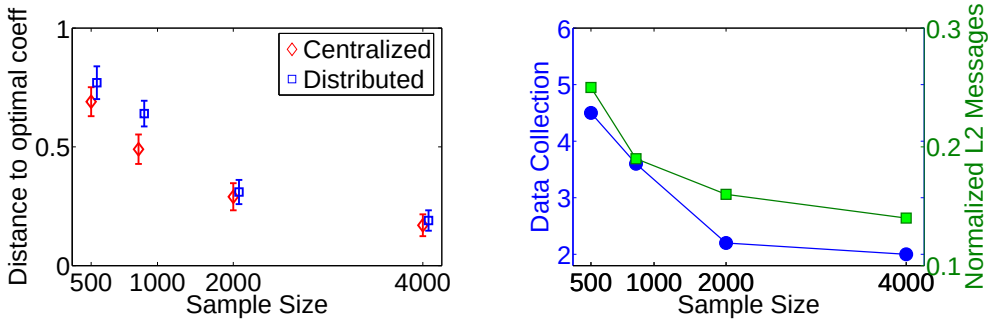
Finally Figures 12(a) and 12(b) demonstrate the effect on quality of the regression model built and the cost incurred as the for building a model of the form $x_3 = a_0\sqrt{x_0 a_1} + a_1\sqrt{x_1 a_0}$. As shown here, the quality improves and the cost decreases as the sample size is increased.

To sum everything up, the regression computation algorithm offers excellent accuracy and low monitoring cost. Also, the number of convergecast-broadcast rounds is also two times per epoch on an average. We have tested our algorithm on several regression functions and the results are highly satisfactory.




(a) Quality for sinusoidal model (b) Cost for sinusoidal model

Fig. 11. Quality and cost of computing regression coefficients for $x_3 = a_0 \times sin(a_1 + a_2 x_1) + a_1 \times sin(a_2 + a_0 x_2)$.




(a) Quality for model $x_3 = a_0\sqrt{x_0 a_1} + a_1\sqrt{x_1 a_0}$ (b) Cost for model $x_3 = a_0\sqrt{x_0 a_1} + a_1\sqrt{x_1 a_0}$

Fig. 12. Quality and cost of computing regression coefficients for $x_3 = a_0\sqrt{x_0 a_1} + a_1\sqrt{x_1 a_0}$.

## VIII. Conclusions and Future Work

To the best of the authors' knowledge this is one of the first attempts on developing a completely local and asynchronous regression algorithm for P2P systems which maintains the same regression models given all the data to all the peers. The algorithm is suitable for scenarios in which the data is distributed across a large P2P network as it seamlessly handles data changes and node failures. We have performed dynamic experiments with random epoch changes which showed that the algorithm is accurate, efficient and highly scalable. Such algorithms are needed for next generation P2P applications such as P2P bioinformatics, P2P web mining and P2P astronomy using National Virtual Observatories. As a next step, we plan to explore other methods of learning such as support vector machines and decision trees.

## ACKNOWLEDGEMENT

This work was supported by the United States National Science Foundation Grant IIS-0093353 and NASA Grant NNX07AV70G.

## REFERENCES

[1] "Chinook," http://smweb.bcgsc.bc.ca/chinook/index.html.

[2] K. Liu, K. Bhaduri, K. Das, and H. Kargupta, "Client-side Web Mining for Community Formation in Peer-to-Peer Environments," *SIGKDD Explorations*, vol. 8, no. 2, pp. 11–20, December 2006.

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

[4] D. E. Hershberger and H. Kargupta, "Distributed Multivariate Regression Using Wavelet-based Collective Data Mining," *Journal of Parallel and Distributed Computing*, vol. 61, no. 3, pp. 372–400, 2001.

[5] C. Guestrin, P. Bodi, R. Thibau, M. Paski, and S. Madden, "Distributed Regression: an Efficient Framework for Modeling Sensor Network Data," in *Proceedings of IPSN'04*, Berkeley, California, 2004, pp. 1–10.

[6] J. Predd, S. Kulkarni, and H. Poor, "Distributed Kernel Regression: An Algorithm for Training Collaboratively," *arXiv.cs.LG archive*, 2006.

[7] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 2, pp. 123–140, 1996.

[8] J. Friedman, T. Hastie, and R. Tibshirani, "Additive Logistic Regression: a Statistical View of Boosting," Dept. of Statistics, Stanford University, Tech. Rep., 1998.

[9] S. J. Stolfo, A. L. Prodromidis, S. Tselepis, W. Lee, D. W. Fan, and P. K. Chan, "JAM: Java Agents for Meta-Learning over Distributed Databases," in *Proceedings of KDD'97*, Newport Beach, California, 1997, pp. 74–81.

[10] Y. Xing, M. G. Madden, J. Duggan, and G. J. Lyons, "Distributed Regression for Heterogeneous Data Sets," *Lecture Notes in Computer Science*, vol. 2810, pp. 544–553, 2003.

[11] D. Caragea, A. Silvescu, and V. Honavar, "A Framework for Learning from Distributed Data Using Sufficient Statistics and Its Application to Learning Decision Trees," *International Journal of Hybrid Intelligent Systems*, vol. 1, no. 1-2, pp. 80–89, 2004.

[12] C. Giannella, K. Liu, T. Olsen, and H. Kargupta, "Communication Efficient Construction of Deicision Trees Over Heterogeneously Distributed Data," in *Proceedings of ICDM'04*, Brighton, UK, 2004, pp. 67–74.

[13] T. Olsen, "Distributed Decision Tree Learning From Multiple Heterogeneous Data Sources," Master's thesis, University of Maryland, Baltimore County, Baltimore. Maryland, October 2006.

[14] S. Merugu and J. Ghosh, "A Distributed Learning Framework for Heterogeneous Data Sources," in *Proceedings of KDD'05*, 2005, pp. 208–217.

[15] B. Park, R. Ayyagari, and H. Kargupta, "A Fourier Analysis-Based Approach to Learn Classifier from Distributed Heterogeneous Data," in *Proceedings of SDM'01*, Chicago, IL, April 2001.

[16] P. Luo, H. Xiong, K. Lü, and Z. Shi, "Distributed Classification in Peer-to-Peer Networks," in *Proceedings of KDD'07*, 2007, pp. 968–976.

[17] K. Bhaduri, R. Wolff, C. Giannella, and H. Kargupta, "Distributed Decision Tree Induction in Peer-to-Peer Systems," *Statistical Analysis and Data Mining Journal (accepted in press)*, 2008.

[18] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta, "Distributed Data Mining in Peer-to-Peer Networks," *IEEE Internet Computing*, vol. 10, no. 4, pp. 18–26, 2006.

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

[20] S. Mukherjee and H. Kargupta, "Distributed Probabilistic Inferencing in Sensor Networks using Variational Approximation," *Journal of Parallel and Distributed Computing*, vol. 68, no. 1, pp. 78–92, 2008.

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

[22] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip Algorithms: Design, Analysis and Applications," in *Proceddings of INFOCOMM'05*, Miami, March 2005, pp. 1653–1664.

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

[24] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[25] J. V. Neumann, *Theory of Self-Reproducing Automata*. Champaign, IL, USA: University of Illinois Press, 1966.

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

[27] N. Linial, "Locality in Distributed Graph Algorithms," *SIAM Journal of Computing*, vol. 21, pp. 193–201, 1992.

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

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

[30] F. Kuhn, T. Moscibroda, and R. Wattenhofer, "What Cannot Be Computed Locally!" in *Proceedings of PODC'04*, St. John's, Newfoundland, Canada, 2004, pp. 300–309.

[31] R. Wolff and A. Schuster, "Association Rule Mining in Peer-to-Peer Systems," *IEEE Transactions on Systems, Man and Cybernetics - Part B*, vol. 34, no. 6, pp. 2426 – 2438, December 2004.

[32] D. Krivitski, A. Schuster, and R. Wolff, "Local Hill Climbing in Sensor Networks," in *Proceedings of DMSN workshop, in conjunction with SDM'05*, Newport Beach, California, April 2005.

[33] R. Wolff, K. Bhaduri, and H. Kargupta, "Local L2 Thresholding Based Data Mining in Peer-to-Peer Systems," in *Proceedings of SDM'06*, Bethesda, MD, 2006, pp. 430–441.

[34] 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.

[35] Y. Birk, I. Keidar, L. Liss, A. Schuster, and R. Wolff, "Veracity Radius - Capturing the Locality of Distributed Computations," in *Proceedings of PODC '06*, Colorado, Denver, 2006, pp. 102–111.

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

[37] J. Garcia-Luna-Aceves and S. Murthy, "A Path-Finding Algorithm for Loop-Free Routing," *IEEE Transactions on Networking*, vol. 5, no. 1, pp. 148–160, 1997.

[38] N. Li, J. Hou, and L. Sha, "Design and Analysis of an MST-Based Topology Control Algorithm," *IEEE Transactions on Wireless Communications*, vol. 4, no. 3, pp. 1195–1205, 2005.

[39] K. Bhaduri, "Efficient local algorithms for distributed data mining in large scale peer to peer environments: A deterministic approach," Ph.D. dissertation, University of Maryland, Baltimore County, March 2008.

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

[41] D. S. Scherber and H. S. Papadopoulos, "Distributed Computation of Averages Over ad hoc Networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 4, pp. 776–787, 2005.

[42] "DDMT," http://www.umbc.edu/ddm/Sftware/DDMT/.

[43] "BRITE," http://www.cs.bu.edu/brite/.

[44] J. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, pp. 308–313, 1965.

**Input**: $\delta^i$, $\Gamma^i, L, \beta$
**Output**: if $\Delta^i \geq 0$ then 1 else 0
**Local variables:** $\forall P_j \in \Gamma^i : \delta^{j,i}, \delta^{i,j}$
**Definitions:** $\Delta^i = \delta^i + \sum_{P_j \in \Gamma^i} \delta^{j,i}$, $\Delta^{i,j} = \delta^{i,j} + \delta^{j,i}$
**Initialization:**
**begin**
  **forall** $P_j \in \Gamma^i$ **do**
    $\delta^{i,j} = \delta^{j,i} = 0$;
    SendMessage($P_j$);
  **end**
**end**
**if** MessageRecvd$(P_j, \delta)$ **then** $\delta^{j,i} \leftarrow \delta$;
**if** PeerFailure$(P_j \in \Gamma^i)$ **then** $\Gamma^i \leftarrow \Gamma^i \setminus \{P_j\}$;
**if** AddNeighbor$(P_j \in \Gamma^i)$ **then** $\Gamma^i \leftarrow \Gamma^i \cup \{P_j\}$;
**if** $\Gamma^i, \delta^i$ *changes or MessageRecvd* **then** call **OnChange**();
Function ***OnChange()***
**begin**
  **forall** $P_j \in \Gamma^i$ **do**
    **if** $(\Delta^{i,j} \geq 0 \wedge \Delta^{i,j} > \Delta^i) \vee (\Delta^{i,j} < 0 \wedge \Delta^{i,j} < \Delta^i)$ **then**
      call **SendMessage**($P_j$);
    **end**
  **end**
**end**
Function ***SendMessage(***$P_j$***)***
**begin**
  **if** $time\,() - last\_message \geq L$ **then**
    $\delta^{i,j} \leftarrow (\beta\Delta^i - \delta^{j,i})$;
    $last\_message \leftarrow time\,()$;
    Send $\langle \delta^{i,j} \rangle$ to $P_j$;
  **end**
  **else**
    Wait $L - (time\,() - last\_message)$ time units;
    Call **OnChange**();
  **end**
**end**

**Algorithm 2**: Local Majority Vote

**Input**: $\epsilon$, $C_\omega$, $S_i$, $\Gamma_i$, $L$, $\widehat{f}$ and $\tau$
**Output**: $\widehat{f}$ such that $\mathcal{E}^{\mathcal{G}} < \epsilon$
**Initialization**
**begin**
    Initialize vectors;
    *MsgType* = MessageRecvdFrom($P_j$);
**end**
**if** $MsgType = Monitoring\_Msg$ **then** Pass Message to Monitoring Algorithm;
**if** $MsgType = New\_Model\_Msg$ **then**
    Update $\widehat{f}$;
    Forward new $\widehat{f}$ to all neighbors;
    Datasent=false;
    Restart Monitoring Algorithm with new $\widehat{f}$;
**end**
**if** $MsgType = Dataset\_Msg$ **then**
    **if** Received from all but one neighbor **then**
        $flag$=Output Monitoring Algorithm();
        **if** $Datasent = false$ *and* $flag = 1$ **then**
            **if** DataAlert stable for $\tau$ time **then**
                $D$=Sample($S_i$, $Recvd\_Dataset$);
                $Datasent$=true;
                Send $D$ to remaining neighbor
            **end**
            **else** *DataAlert*=CurrentTime;
        **end**
    **end**
    **if** Received from all neighbors **then**
        $D$=Sample($S_i$, $Recvd\_Dataset$);
        $\widehat{f}$=Regression($D$);
        Forward new $\widehat{f}$ to all neighbors;
        *Datasent*=false;
        Restart Monitoring Algorithm with new $\widehat{f}$;
    **end**
**end**
**if** $S_i$, $\Gamma_i$ *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 3**: P2P Regression Algorithm.