Scalable, Asynchronous, Distributed Eigen-Monitoring of Astronomy Data Streams

Kanishka Bhaduri[*], Kamalika Das[†], Kirk Borne[‡], Chris Giannella[§], Tushar Mahule[¶], Hillol Kargupta[¶‖]

[*]Mission Critical Technologies Inc.,

NASA Ames Research Center, MS 269-1, Moffett Field, CA-94035

Email:Kanishka.Bhaduri-1@nasa.gov

[†]Stinger Ghaffarian Technologies Inc.,

NASA Ames Research Center, MS 269-3, Moffett Field, CA-94035

Email:Kamalika.Das@nasa.gov

[‡]Computational and Data Sciences Dept., GMU, VA-22030

Email:kborne@gmu.edu

[§]The MITRE Corporation

300 Sentinel Dr. Suite 600, Annapolis Junction MD 20701

Email:cgiannel@acm.org

[¶]CSEE Department, UMBC

1000 Hilltop Circle, Baltimore, Maryland, 21250, USA

Email:{tusharm1,hillol}@cs.umbc.edu

[‖]AGNIK, LLC., Columbia, MD, USA

A shorter version of this paper was published in SIAM Data Mining Conference 2009

Abstract

In this paper we develop a distributed algorithm for monitoring the principal components (PCs) for next generation of astronomy petascale data pipelines such as the Large Synoptic Survey Telescopes (LSST). This telescope will take repeat images of the night sky every 20 seconds, thereby generating 30 terabytes of calibrated imagery every night that will need to be co-analyzed with other astronomical data stored at different locations around the world. Event detection, classification and isolation in such data sets may provide useful insights to unique astronomical phenomenon displaying astrophysically significant variations: quasars, supernovae, variable stars, and potentially hazardous asteroids. However, performing such data mining tasks is a challenging problem for such high-throughput distributed data streams. In this paper we propose a highly scalable and distributed asynchronous algorithm for monitoring the principal components (PC) of such dynamic data streams and discuss a prototype web-based system PADMINI (Peer to Peer Astronomy Data Mining) which implements this algorithm for use by the astronomers. We demonstrate the algorithm on a large set of distributed astronomical data to accomplish well-known astronomy tasks such as measuring variations in the fundamental plane of galaxy parameters. The proposed algorithm is provably correct (*i.e.* converges to the correct PCs without centralizing any data) and can seamlessly handle changes to the data or the network. Real experiments performed on Sloan Digital Sky Survey (SDSS) catalogue data show the effectiveness of the algorithm.

I. INTRODUCTION

Data mining is playing an increasingly important role in astronomy research [25][9][4] involving very large sky surveys such as Sloan Digital Sky Survey SDSS and the 2-Micron All-Sky Survey 2MASS. These sky-surveys are offering a new way to study and analyze the behavior of the astronomical objects. The next generation of sky-surveys are poised to take a step further by incorporating sensors that will stream in large volume of data at a high rate. For example, the Large Synoptic Survey Telescopes (LSST) will take repeated images of the night sky every 20 seconds. This will generate 30 terabytes of calibrated imagery every night that will need to be co-analyzed with other astronomical data stored at different locations around the world. Event identification and classification in such data sets may provide useful insights to unique astronomical phenomenon displaying astrophysically significant variations: quasars, supernovae, variable stars, and potentially hazardous asteroids. Analyzing such high-throughput data streams would require large distributed computing environments for offering

scalable performance. The knowledge discovery potential of these future massive data streams will not be achieved unless novel data mining algorithms are developed to handle decentralized petascale data flows, often from multiple distributed sensors (data producers) and archives (data providers). Several distributed computing frameworks such as [16], [22], [23], [19], and [14] are being developed for such applications. We need distributed data mining algorithms that can operate on such distributed computing environments. These algorithms should be highly scalable, be able to provide good accuracy and should have a low communication overhead.

This paper considers the problem of monitoring the spectral properties of data streams in a distributed environment. It offers an asynchronous, communication-efficient distributed eigen monitoring algorithm for monitoring the principle components (PCs) of dynamic astronomical data streams. It particularly considers an important problem in astronomy regarding the variation of fundamental plane structure of galaxies with respect to spatial galactic density and demonstrates the power of DDM algorithms using this example application. This paper presents the algorithm, analytical findings, and results from experiments performed using currently available astronomy data sets from virtual observatories. Our distributed algorithm is a first step in analyzing the astronomy data arriving from such high throughput data streams of the future. The specific contributions of this paper can be summarized 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 eigen analysis in distributed data streams
- Based on data sets downloaded from astronomy catalogues such as SDSS and 2MASS, we demonstrate how the galactic fundamental plane structure varies with difference in galactic density
- We discuss the architecture, workflow and deployment of an entirely web-based P2P astronomy data mining prototype (PADMINI) that allows astronomers to perform event detection and analysis using P2P data mining technology

Section II describes the astronomy problem. Section III presents the related work. Section IV describes the centralized version of the problem while Section V models the distributed version and explains our distributed the eigenstate monitoring algorithm. Section VI presents the experimental results followed by a web-based astronomy PADMINI system in Section VII.

Finally, Section VIII concludes this paper.

II. Astronomy data stream challenge problems

When the LSST astronomy project becomes operational within the next decade, it will pose enormous petascale data challenges. This telescope will repeatedly take images of the night sky every 20 seconds, throughout every night, for 10 years. Each image will consist of 3 gigapixels, yielding 6 gigabytes of raw imagery every 20 seconds and nearly 30 terabytes of calibrated imagery every night. From this "cosmic cinematography", a new vision of the night sky will emerge – a vision of the temporal domain – a ten-year time series (movie) of the Universe. Astronomers will monitor these repeat images night after night, for 10 years, for everything that has changed – changes in position and intensity (flux) will be monitored, detected, measured, and reported. For those temporal variations that are novel, unexpected, previously unknown, or outside the bounds of our existing classification schemes, astronomers will want to know quickly if such an event has occurred. This event alert notification must necessarily include as much information as possible to help the astronomers around the world to prioritize their response to each time-critical event. This information packet will include a probabilistic classification of the event, with some measure of the confidence of the classification. What makes the LSST so incredibly beyond current projects is that most time-domain sky surveys today detect 5-10 events per week; LSST will detect 10 to 100 thousand events per night! Without good classification information in those alert packets, and hence without some means with which to prioritize the huge number of events, the astronomy community would consequently be buried in the data deluge and will miss some of the greatest astronomical discoveries of the next 20 years *e.g.* even the next "killer asteroid" heading for Earth.

To solve the astronomers' massive event classification problem, a collection of high-throughput monitoring and event detection algorithms will be needed. These algorithms will need to access distributed astronomical databases worldwide to correlate with each of those 100,000 nightly events, in order to model, classify, and prioritize each event correctly and rapidly. One known category of temporally varying astronomical object is a variable star. There are dozens of different well known classes of variable stars, and there are hundreds (even thousands) of known examples of these classes. These stars are not "interesting" in the sense that they should not produce alerts, even though they are changing in brightness from hour to hour, night to night, week to week –

their variability is known, well studied, and well characterized already. However, if one of these stars' class of variability were to change, that would be extremely interesting and be a signal that some very exotic astrophysical processes are involved. Astronomers will definitely want to be notified promptly (with an alert) of these types of variations. One way of characterizing this variation is by studying changes in the Fourier components (eigenvectors) of the temporal flux curve which astronomers call "the light curve".

This problem has several interesting data challenge characteristics: (1) the data streaming rate is enormous (6 gigabytes every 20 seconds); (2) there are roughly 100 million astronomical objects in each of these images, all of which need to monitored for change (*i.e.*, a new variable object, or a known variable object with a new class of variability); (3) 10 to 100 thousand "new" events will be detected each and every night for 10 years; and (4) distributed data collections accessed through the Virtual Astronomy Observatory's worldwide distribution of databases and data repositories will need to correlated and mined in conjunction with each new variable object's data from LSST, in order to provide the best classification models and probabilities, and thus to generate the most informed alert notification messages.

Astronomers cannot wait until the year 2016 when LSST begins its sky survey operations for new algorithms to begin being researched. Those algorithms should be able to analyze the data *in-situ*, without the costly need for centralizing all of it for analysis at each time point. Furthermore, the distributed mining algorithms will need to be robust, scalable, and validated already at that time. So, it is imperative to begin now to research, test, and validate such data mining paradigms through experiments that replicate the expected conditions of the LSST sky survey. Consequently, we have chosen an astronomical research problem that is both scientifically valid (*i.e.*, a real astronomy research problem today) and that parallels the eigenvector monitoring problem that we have described above. We have chosen to study the principal components of galaxy parameters as a function of an independent variable, similar to the temporal dynamic stream mining described above. In our current experiments, the independent variable is not the time dimension, but local galaxy density. We specifically investigate this problem because it is scientifically current and interesting, thereby producing new astronomical research results, and also because it performs tests of our algorithms specifically on the same types of distributed databases that will be used in the future LSST event classification problems

The class of elliptical galaxies has been known for 20 years to show dimension reduction

among a subset of physical attributes, such that the 3-dimensional distribution of three of those astrophysical parameters reduce to a 2-dimensional plane. The normal to that plane represents the principal eigenvector of the distribution, and it is found that the first two principal components capture significantly more than 90% of the variance among those 3 parameters.

By analyzing existing large astronomy databases (such as the Sloan Digital Sky Survey SDSS and the 2-Micron All-Sky Survey 2MASS), we have generated a very large data set of galaxies. Each galaxy in this large data set was then assigned (labeled with) a new "local galaxy density" attribute, calculated through a volumetric Voronoi tessellation of the total galaxy distribution in space. Then the entire galaxy data set was horizontally partitioned across several dozen partitions as a function of our independent variable: the local galaxy density.

As a result, we have been able to study eigenvector changes of the fundamental plane of elliptical galaxies as a function of density. Computing these eigenvectors for a very large number of galaxies, one density bin at a time, in a distributed environment, thus mimics the future LSST dynamic data stream mining (eigenvector change) challenge problem described earlier. In addition, this galaxy problem actually has uncovered some new astrophysical results: we find that the variance captured in the first 2 principal components increases systematically from low-density regions of space to high-density regions of space, and we find that the direction of the principal eigenvector also drifts systematically in the 3-dimensional parameter space from low-density regions to the highest-density regions. However, since the focus of this paper is on the distributed algorithms themselves and not the discovery of new astrophysical results, we leave a detailed discussion of them to another paper.

III. Related work

The work related to this area of research can broadly be subdivided into data analysis for large scientific data repository and distributed data mining in a dynamic networks of nodes. We discuss each of them in the following two sections.

A. Analysis of large scientific data collections

The U.S. National Virtual Observatory (NVO) [35] is a large scale effort to develop an information technology infrastructure enabling easy and robust access to distributed astronomical archives. It will provide services for users to search and gather data across multiple archives

and will provide some basic statistical analysis and visualization functions. The International Virtual Observatory Alliance (IVOA) [27] is the international steering body that federates the work of about two dozen national VOs across the world (including the NVO in the US). The IVOA oversees this large-scale effort to develop an IT infrastructure enabling easy and robust access to distributed astronomical archives worldwide.

There are several instances in the astronomy and space sciences research communities where data mining is being applied to large data collections [16][13][2]. Another recent area of research is distributed data mining [30][28] which deals with the problem of data analysis in environments with distributed data, computing nodes, and users. Distributed eigen-analysis and outlier detection algorithms have been developed for analyzing astronomy data stored at different locations by Dutta *et al.*[20]. Kargupta *et al.* [29] have developed a technique for performing distributed principal component analysis by first projecting the local data along its principal components and then centralizing the projected data. In both these cases, the data is distributed vertically (different full attribute columns reside at different sites), while in this paper, the data is distributed horizontally (different data tuple sets reside at different sites). Moreover, none of the above efforts address the problem of analyzing rapidly changing astronomy data streams.

B. Data analysis in large dynamic networks

There is a significant amount of recent research considering data analysis in large-scale dynamic networks. Since efficient data analysis algorithms can often be developed based on efficient primitives, approaches have been developed for computing basic operations (*e.g.* average, sum, max, random sampling) on large-scale, dynamic networks. Kempe *et al.* [31] and Boyd *et al.* [10] developed gossip based randomized algorithms. These approaches used an epidemic model of computation. Bawa *et al.* [5] developed an approach based on probabilistic counting. In addition, techniques have been developed for addressing more complex data mining/data problems over large-scale dynamic networks: association rule mining [39], facility location [32], outlier detection [11], decision tree induction [8], ensemble classification [33], support vector machine-based classification [1], k-means clustering [15], top-k query processing [3].

A related line of research concerns the monitoring of various kinds of data models over large numbers of data streams. Sharfman *et al.* [37] develop an algorithm for monitoring arbitrary threshold functions over distributed data streams. And, most relevant to this paper, Wolff *et al.*

[38] developed an algorithm for monitoring the L2 norm. We use this technique to monitor eigenstates of the fundamental plane concerning elliptical galaxies. The formulation and experimental results presented in this paper are new and do not appear in [38].

Huang *et al.* [26] consider the problem of detecting network-wide volume anomalies via thresholding the length of a data vector (representing current network volume) projected onto a subspace closely related to the dominant principal component subspace of past network volume data vectors. Unlike us, these authors consider the analysis of a vertically distributed data set. Each network node holds a sliding window stream of numbers (representing volume through it with time) and the network-wide volume is represented as a matrix with each column a node stream. Because of the difference in data distribution (vertical vs. horizontal), their approach is not applicable to our problem. We assume that each node is receiving a stream of tuples and the network-wide dataset is matrix formed by the union of all nodes' currently held tuples (each node holds a collection of *rows* of the matrix rather than a single *column* as considered by Huang).

In the next few sections we first discuss our analysis for identifying the fundamental plane of elliptical galaxies, and then show how the same computation can be carried out if the data is stored at multiple locations.

IV. Centralized Principal Components Analysis for the Fundamental Plane computation

The identification of certain correlations among parameters has lead to important discoveries in astronomy. For example, the class of elliptical and spiral galaxies (including dwarfs) have been found to occupy a 2D space inside a 3D space of observed parameters — radius, mean surface brightness and velocity dispersion. From this 3D space of observed parameters, the 2D plane can be derived by projecting the data on the top 2 eigenvectors of the covariance matrix *i.e.* performing a principal component analysis (PCA) of the covariance matrix of the data. This 2D plane has been referred to as the Fundamental Plane [21]. We study the variation of this fundamental plane with the density of each galaxy derived from location and other observed parameters.

A. Data preparation

For identifying the variability of fundamental plane on the basis of galactic densities, we have used the SDSS and 2MASS data sets available individually through the NVO. Since galactic density is not observed by the NVOs, we have cross-matched the two data sets and computed the densities based on other property values the details of which we describe next.

We create a large, aggregate data set by downloading the 2MASS XSC extended source catalog (`http://irsa.ipac.caltech.edu/applications/Gator/`) for the entire sky and cross-match it against the SDSS catalog using the SDSS Crossid tool (`http://cas.sdss.org/astro/en/tools/crossid/upload.asp`) such that we select all unique attributes from the *PhotoObjAll* and *SpecObjAll* tables as well as the *photozd1* attribute from the *Photoz2* table which is an estimated redshift value. We filter the data based on the SDSS identified type to remove all non-galaxy tuples. We then filter the data again based on reasonable redshift (actual or estimated) values between $0.003 \leq z \leq 0.300$.

Next, we create a new attribute, local galactic density to quantify the proximity of nearby galaxies to a given galaxy (this attribute has strong astrophysical significance). We transformed the attributes cx, cy, cz (unit vectors), z, and $photozd1$ to 3D Euclidean coordinates

$$(X, Y, Z) = (Distance \times cx, Distance \times cy, Distance \times cz)$$

where $Distance = 2 \times \left[1 - \frac{1}{\sqrt{(1+redshift)}}\right]$. We finally use these Cartesian coordinates to compute the Delaunay Tessellation [18] of each point (galaxy) in 3D space. The local galactic density of a given galaxy i is computed using the Delaunay Tessellation Field Estimator (DTE) formulation [36]:

$$den(i) = \frac{4}{vol(i)}$$

where $vol(i)$ denotes the volume of the Delaunay cell containing galaxy i. A small number of galaxies have undefined $den(i)$ because they are on the boundary and have $vol(i) = \infty$. These galaxies are dropped from our calculations.

B. Binning and PCA

The astronomy question that we want to address here is whether the fundamental plane structure of galaxies in low density regions differ from that of galaxies in high density regions.

For this, we take the above data set containing 155650 tuples and associate with each tuple, a measure of its local galactic density. Our final aggregated data set has the following attributes from SDSS: Petrosian I band angular effective radius (*Iaer*), redshift (*rs*), and velocity dispersion (*vd*); and has the following attribute from 2MASS: K band mean surface brightness (*Kmsb*). We produce a new attribute, logarithm Petrosian I band effective radius (*log(Ier)*), as *log(Iaer*rs)* and a new attribute, logarithm velocity dispersion (*log(vd)*), by applying the logarithm to *vd*. We finally append the galactic density (*cellDensity*) associated with each of the tuples as the last attribute of out aggregated data set. We divide the tuples into 30 bins based on increasing cell density, such that there are equal number of tuples in each bin. For each bin we carry out the fundamental plane calculation or principal component analysis and observe that the percent of variance captured by the first two PCs is very high. This implies that the galaxies can be represented by the plane defined by the first two eigen vectors. It is also observed that this percentage increases for bins with higher mean galactic density. We report these results in Section VI.

As discussed earlier, analysis of very large astronomy catalogs can pose serious scalability issues, especially when considering streaming data from multiple sources. In the next section we describe a distributed architecture for addressing these issues and then show how the centralized eigen analysis of the covariance matrix can be formulated as a distributed computation and solved in a communication efficient manner.

V. Distributed Principal Component Analysis

When resources become a constraint for doing data mining on massive data sets such as astronomical catalogs, distributed data mining provides a communication efficient solution. For the problem discussed in the last section, we can formulate a distributed architecture where after cross matching the data using a centralized cross matching tool, we can store the meta data information in a central location. Such a service-oriented architecture would facilitate astronomers to query multiple databases and do data mining on large data sets without downloading the data to their local computing resources. The data set is downloaded in parts at a number of computing nodes (that are either dedicated computers connected through communication channels or part of a large grid) based on the meta data information maintained at the central server site. In such a computational environment, distributed data mining algorithms can run in the background

seamlessly for providing fast and efficient solutions to the astronomers by distributing the task among a number of nodes. Figure 1 represents one such architecture in which the user submits jobs through the web server and the DDM server will execute these jobs using the underlying P2P architecture.

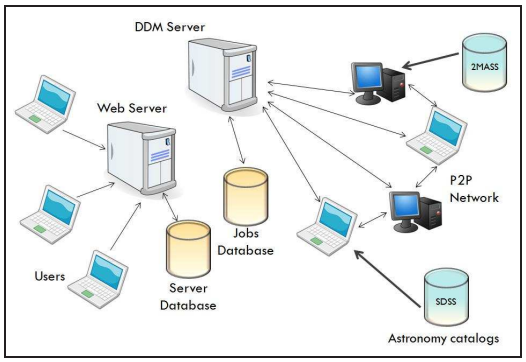


Fig. 1. System diagram showing the different components.

A. Notations

Let $V = \{P_1, \ldots, P_n\}$ be a set of nodes 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. Additionally, at any time, P_i is given a time-varying data matrix $\mathcal{M}_i$ of size $|\mathcal{M}_i|$ where the rows correspond to the instances and the columns correspond to attributes or features. Mathematically, $\mathcal{M}_i = [\overrightarrow{x_{i,1}} \quad \overrightarrow{x_{i,2}} \ldots]^{\mathrm{T}}$, where each $\overrightarrow{x_{i,\ell}} = [x_{i,\ell 1} \quad x_{i,\ell 2} \ldots x_{i,\ell d}] \in \mathbb{R}^d$ is a row (tuple). The covariance matrix of the data at node P_i, denoted by $\mathcal{C}_i$, is the matrix whose (i,j)-th entry corresponds to the covariance between the i-th and j-th feature (column) of $\mathcal{M}_i$. The global data set of all the nodes is $\mathcal{G} = \bigcup_{i=1}^{n} \mathcal{M}_i$ and the global covariance matrix is $\mathcal{C}$. Let $\overrightarrow{V}$, Θ and $\overrightarrow{\mu}$ denote the eigenvector (assumed to be of length one), eigenvalue and mean of the global data respectively. Throughout this discussion we have dropped the explicit time subscript.

B. Problem formulation

The problem that we want to solve in this paper can be stated as follows:

Definition 5.1 (Problem Statement): Given a time-varying data set $\mathcal{M}_i$ at each node, maintain an up-to-date set of eigenvectors ($\overrightarrow{V}$) and eigenvalues (Θ) of the global covariance matrix $\mathcal{C}$ *i.e.*

find V and Θ such that

$$\mathcal{C}\overrightarrow{V} = \Theta\overrightarrow{V}$$

In our scenario, since the data is constantly changing, we relax this requirement and use the following as an admissible solution.

Definition 5.2 (Relaxed Problem Statement): Given a time-varying data set $\mathcal{M}_i$ at each node, maintain an up-to-date set of eigenvectors ($\overrightarrow{V}$) and eigenvalues (Θ) of the global covariance matrix $\mathcal{C}$ such that,

$$\left\|\mathcal{C}\overrightarrow{V} - \Theta\overrightarrow{V}\right\| < \epsilon$$

where ϵ is a user chosen parameter denoting the error threshold.

C. Distributed PCA monitoring

One way of keeping the model up-to-date is by periodically rebuilding the model. However, this wastes resources if the data is stationary. Alternatively, one may risk model inaccuracy if the period of recomputation is too long and the data changes in between.

In this work, we take a different approach. Starting with an arbitrary model at each node, we propose an algorithm which raises an alert whenever the global data of the nodes can no longer fit this model. If the data has changed enough, we use a feedback loop to collect data from the network (using convergecast), rebuild a new model and then distribute this new model to all the nodes to be again tracked by the peers against the current data. Below, we reduce the problem of monitoring the eigenvectors and eigenvalues to checking if a local vector at each peer is inside a circle of radius ϵ.

Note that, if all the columns of $\mathcal{G}$ are mean reduced (using the global mean) by the respective columns, *i.e.* the mean of each column is subtracted from each entry of that column, the covariance matrix is decomposable: $\mathcal{C} = \sum_{i=1}^{n} \mathcal{C}_i$. With abuse of symbols, let $\mathcal{G}$ and $\mathcal{M}_i$ denote

the mean reduced versions of the variables themselves. Then,

$$\begin{aligned}
\mathcal{C} = \frac{1}{\sum_i^n |\mathcal{M}_i|}\mathcal{G}^T\mathcal{G} &= \frac{1}{\sum_i^n |\mathcal{M}_i|}\begin{pmatrix}\mathcal{M}_1\\ \mathcal{M}_2\\ \vdots\\ \mathcal{M}_n\end{pmatrix}^T\begin{pmatrix}\mathcal{M}_1\\ \mathcal{M}_2\\ \vdots\\ \mathcal{M}_n\end{pmatrix}\\
&= \frac{1}{\sum_i^n |\mathcal{M}_i|}\begin{pmatrix}\mathcal{M}_1^T & \mathcal{M}_2^T \dots & \mathcal{M}_n^T\end{pmatrix}\begin{pmatrix}\mathcal{M}_1\\ \mathcal{M}_2\\ \vdots\\ \mathcal{M}_n\end{pmatrix}\\
&= \frac{1}{\sum_i^n |\mathcal{M}_i|}\sum_{i=1}^{n}\mathcal{M}_i^T\mathcal{M}_i\\
&= \frac{1}{\sum_i^n |\mathcal{M}_i|}\sum_{i=1}^{n}\mathcal{C}_i \qquad (1)
\end{aligned}$$

Thus, for horizontally partitioned mean reduced data distributed among n nodes, the covariance matrix is completely decomposable. Assuming that each peer is provided with an initial estimate of $\overrightarrow{V}$ (with $||\overrightarrow{V}|| = 1$) and Θ, the eigen monitoring instance (denoted by I_1) can be reformulated as:

$$\begin{aligned}
\left\| \mathcal{C}\overrightarrow{V} - \Theta\overrightarrow{V}\right\| < \epsilon_1 &\Leftrightarrow \left\|\left(\frac{1}{\sum_i^n |\mathcal{M}_i|}\sum_i^n \mathcal{C}_i\right)\overrightarrow{V} - \Theta\overrightarrow{V}\right\| < \epsilon_1\\
&\Leftrightarrow \left\|\frac{1}{\sum_i^n |\mathcal{M}_i|}\sum_i^n\left[\mathcal{C}_i\overrightarrow{V} - \Theta\overrightarrow{V}|\mathcal{M}_i|\right]\right\| < \epsilon_1\\
&\Leftrightarrow \left\|\sum_i^n \frac{|\mathcal{M}_i|}{\sum_i^n |\mathcal{M}_i|}\left[\frac{\mathcal{C}_i\overrightarrow{V}}{|\mathcal{M}_i|} - \Theta\overrightarrow{V}\right]\right\| < \epsilon_1\\
&\Leftrightarrow \left\|\sum_i^n \left(\frac{|\mathcal{M}_i|}{\sum_i^n |\mathcal{M}_i|}\right)\left[I_1.\overrightarrow{\mathcal{E}_i}\right]\right\| < \epsilon_1 \qquad (2)
\end{aligned}$$

where $I_1.\overrightarrow{\mathcal{E}_i}$ is a local error vector at node P_i (based on $\mathcal{M}_i$, $\overrightarrow{V}$ and Θ) defined as $I_1.\overrightarrow{\mathcal{E}_i} = \frac{\mathcal{C}_i\overrightarrow{V}}{|\mathcal{M}_i|} - \Theta\overrightarrow{V}$. Let $I_1.\overrightarrow{\mathcal{E}}^G = \sum_i^n \left(\frac{|\mathcal{M}_i|}{\sum_i^n |\mathcal{M}_i|}\right)\left[I_1.\overrightarrow{\mathcal{E}_i}\right]$ denote the convex combination of $I_1.\overrightarrow{\mathcal{E}_i}$'s. Checking if the norm of $I_1.\overrightarrow{\mathcal{E}_i}$ is less than ϵ is equivalent to checking if the vector $I_1.\overrightarrow{\mathcal{E}_i}$ is inside a sphere of radius ϵ_1. Now if each peer determines that their own vector $I_1.\overrightarrow{\mathcal{E}_i}$ is inside

the sphere, then so is their convex combination $I_1.\overrightarrow{\mathcal{E}}^G$. This is the crux of the idea used in developing the distributed algorithm. However, this argument falls apart when the vectors are outside the sphere. To circumvent this problem and apply the same methodology, the area outside the circle is approximated by a set of tangents to the sphere. As before, if all peer's $I_1.\overrightarrow{\mathcal{E}_i}$ lies in the same hyperplane, $I_1.\overrightarrow{\mathcal{E}}^G$ will also lie there. This paves the way for the distributed algorithm which is discussed next.

Note that, in the above formulation we have assumed that the data is mean shifted. In a dynamic setting, it may be expensive to recompute the mean at each time step. Given an initial estimate of the mean $\overrightarrow{\mu}$ to all the peers (may be a random choice), we set up another monitoring instance I_2 for checking if the (column wise) average vector over all peers exceeds a threshold ϵ_2:

$$\begin{aligned}
\left\| \frac{1}{\sum_{i=1}^{n} \mathcal{M}_i} \sum_{i=1}^{n} \sum_{j=1}^{|\mathcal{M}_i|} \overrightarrow{x_{i,j}} - \overrightarrow{\mu} \right\| < \epsilon_2 \quad &\Leftrightarrow \quad \left\| \frac{1}{\sum_{i=1}^{n} \mathcal{M}_i} \sum_{i=1}^{n} \left[\sum_{j=1}^{|\mathcal{M}_i|} \overrightarrow{x_{i,j}} - \overrightarrow{\mu} |\mathcal{M}_i| \right] \right\| < \epsilon_2 \\
&\Leftrightarrow \quad \left\| \sum_{i=1}^{n} \frac{|\mathcal{M}_i|}{\sum_{i=1}^{n} \mathcal{M}_i} \left[\frac{\sum_{j=1}^{|\mathcal{M}_i|} \overrightarrow{x_{i,j}}}{|\mathcal{M}_i|} - \overrightarrow{\mu} \right] \right\| < \epsilon_2 \\
&\Leftrightarrow \quad \left\| \sum_{i=1}^{n} \left(\frac{|\mathcal{M}_i|}{\sum_{i=1}^{n} \mathcal{M}_i} \right) \left[I_2.\overrightarrow{\mathcal{E}_i} \right] \right\| < \epsilon_2 \qquad (3)
\end{aligned}$$

where, as before, $I_2.\overrightarrow{\mathcal{E}_i} = \left(\frac{\sum_{j=1}^{|\mathcal{M}_i|} \overrightarrow{x_{i,j}}}{|\mathcal{M}_i|} - \overrightarrow{\mu} \right)$ is the local vector and $I_2.\overrightarrow{\mathcal{E}^G} = \sum_{i=1}^{n} \left(\frac{|\mathcal{M}_i|}{\sum_{i=1}^{n} \mathcal{M}_i} \right) \left[I_2.\overrightarrow{\mathcal{E}_i} \right]$ is a convex combination of $I_2.\overrightarrow{\mathcal{E}_i}$-s. The same convex methodology for checking inside and outside of the circle can be applied here.

Satisfying the relaxed problem statement: In the appendix, we show that if the bounds (2) and (3) hold, then the problem statement in the above definition holds with $\epsilon = \epsilon_1 + \epsilon_2^2$.

D. Notations and thresholding criterion

In our algorithm, each node sends messages to its immediate neighbors to converge to a globally correct solution. There are three kinds of messages that can be transmitted: (i) *monitoring* messages which are used by the algorithm to check if the model is up-to-date, (ii) *data* messages which are used to sample data for rebuilding a model (convergecast), and (iii) *model* messages which are used to disseminate the newly built model in the entire network (broadcast). Any monitoring message sent by node P_i to P_j contains information that P_i has gathered about

the network which P_j may not know. In our case, the message sent by P_i to P_j consists of a set of vectors or matrix[1] $\mathcal{S}_{i,j}$ with each row corresponding to observations and each column corresponding to features.

We know that if each peer's $I_1.\overrightarrow{\mathcal{E}_i}$ (or $I_2.\overrightarrow{\mathcal{E}_i}$) lies in the same convex region, $I_1.\overrightarrow{\mathcal{E}}^G$ (or $I_2.\overrightarrow{\mathcal{E}}^G$) also lies in the same convex region. Therefore, each peer needs information about the state of its neighbors. The trick is to do this computation without collecting all of the data of all the peers. We define three sufficient statistics on sets of vectors (average vector of the set and the number of points in the set) at each node and for each instance of the monitoring problem separately, based on which a peer can do this thresholding more efficiently. For the rest of the paper, we only discuss the computations with respect to I_1 since the other instance is very similar.

- **Knowledge** $\overrightarrow{\mathcal{K}_i}$: This is all the information that P_i has about the network.
- **Agreement** $\overrightarrow{\mathcal{A}_{i,j}}$: This is what P_i and P_j have in common.
- **Held** $\overrightarrow{\mathcal{H}_{i,j}}$: This is what P_i has not yet communicated to P_j.

We can write

- $|\mathcal{K}_i| = |\mathcal{M}_i| + \sum_{P_j \in \Gamma_i} |\mathcal{S}_{j,i}|$
- $|\mathcal{A}_{i,j}| = |\mathcal{S}_{i,j}| + |\mathcal{S}_{j,i}|$
- $|\mathcal{H}_{i,j}| = |\mathcal{K}_i| - |\mathcal{A}_{i,j}|$

Similarly for the average of the sets we can write,

- $\overrightarrow{\mathcal{K}_i} = \frac{1}{|\mathcal{K}_i|}\left(|\mathcal{M}_i|\,\overrightarrow{\mathcal{E}_i} + \sum_{P_j \in \Gamma_i} |\mathcal{S}_{j,i}|\,\overrightarrow{\mathcal{S}_{j,i}}\right)$
- $\overrightarrow{\mathcal{A}_{i,j}} = \frac{1}{|\mathcal{A}_{i,j}|}\left(|\mathcal{S}_{i,j}|\,\overrightarrow{\mathcal{S}_{i,j}} + |\mathcal{S}_{j,i}|\,\overrightarrow{\mathcal{S}_{j,i}}\right)$
- $\overrightarrow{\mathcal{H}_{i,j}} = \frac{1}{|\mathcal{H}_{i,j}|}\left(|\mathcal{K}_i|\,\overrightarrow{\mathcal{K}_i} - |\mathcal{A}_{i,j}|\,\overrightarrow{\mathcal{A}_{i,j}}\right)$

In this work we assume that the communication takes place over an overlay tree. This is to ensure that vectors sent to a node is never sent back to it to avoid double counting. Interested readers are urged to see [38] and [8] for a discussion of how this assumption can be accommodated or, if desired, removed.

At each peer, we need to check if the local vector $\overrightarrow{\mathcal{K}_i}$ lies in a convex region. To achieve this, we need to split the domain of monitoring function into non-overlapping convex regions. Since

[1]we use them interchangeably here

the monitoring function is L2 norm in $\mathbb{R}^d$, checking if the norm is less than ϵ is equivalent to checking if it is inside the sphere which is a convex region by definition. However, the outside of the sphere is not convex. So we make it convex by drawing tangents to the sphere at arbitrary points. Each of these half spaces is again convex and so the general rule can be applied here. However, the area between the sphere and each of these half spaces is not convex. These small uncovered spaces are known as the *tie* regions. Denoting the area inside the sphere as R_{in} and each of the half spaces as $\{R_{h_1}, R_{h_2}, \dots\}$, the entire set of convex regions covering the space is $C_\omega = \{R_{in}, R_{h_1}, R_{h_2}, \dots\}$. Fig. 2 shows the convex regions in $\mathbb{R}^d$, the tangent lines and the tie region. Given this convex region and the local vectors, we now state a Theorem based on which any peer can stop sending messages and output the correct result.

Theorem 5.1: [38] Let $\overrightarrow{\mathcal{E}^G}$, $\overrightarrow{\mathcal{K}_i}$, $\overrightarrow{\mathcal{A}_{i,j}}$, and $\overrightarrow{\mathcal{H}_{i,j}}$ be as defined in the previous section. Let R be any region in C_ω. If at time t no messages traverse the network, and for each P_i, $\overrightarrow{\mathcal{K}_i} \in R$ and for every $P_j \in \Gamma_i$, $\overrightarrow{\mathcal{A}_{i,j}} \in R$ and either $\overrightarrow{\mathcal{H}_{i,j}} \in R$ or $\mathcal{H}_{i,j} = \emptyset$, then $\overrightarrow{\mathcal{E}^G} \in R$.

Proof: For proof the readers are referred to [38]. ■

Using this theorem, each node can check if $\left\|\overrightarrow{\mathcal{K}_i}\right\| < \epsilon$. If the result holds for every node, then their convex combination $\overrightarrow{\mathcal{E}^G}$ will also be in R. If there is any disagreement, it would be between any two neighbors. In that case, messages will be exchanged and they will converge to the correct result. In either case, eventual global correctness is guaranteed.

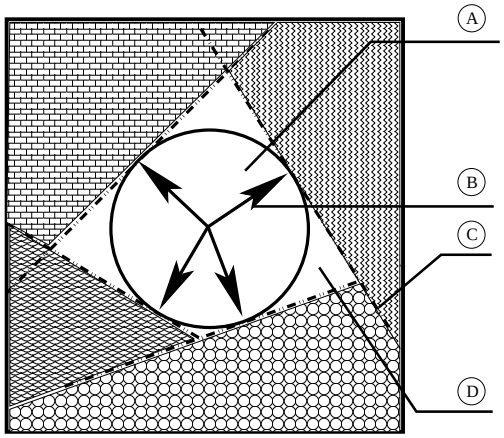


Fig. 2. (A) the area inside an ϵ circle (B) A random vector (C) A tangent defining a half-space (D) The areas between the circle and the union of half-spaces are the tie areas.

E. Algorithm

Both the mean monitoring and the eigenvector monitoring algorithms rely on the results of Theorem 5.1 to output the correct result. For the eigenvector monitoring, the inputs to each node are the eigenvector $\overrightarrow{V}$, the eigenvalue Θ and the error threshold is ϵ_1. From Section V-C, for this monitoring instance, the input is:

- $I_1.\overrightarrow{\mathcal{E}_i} = \frac{\left(\left[\mathcal{M}_i^{\mathrm{T}}\mathcal{M}_i\right].\overrightarrow{V}\right)}{|\mathcal{M}_i|} - \Theta\overrightarrow{V}$
- $I_1.\,|\mathcal{E}_i| = |\mathcal{M}_i|$

Similarly for the mean monitoring algorithm, the inputs are the mean $\overrightarrow{\mu} \in \mathbb{R}^d$ and the error threshold ϵ_2. In this case, each node subtracts the mean $\overrightarrow{\mu}$ from its local average input vector $\overrightarrow{x_{i,j}}$. For this problem instance denoted by I_2, the following are the inputs:

- $I_2.\overrightarrow{\mathcal{E}_i} = \left(\frac{\sum_{j=1}^{|\mathcal{M}_i|}\overrightarrow{x_{i,j}}}{|\mathcal{M}_i|} - \overrightarrow{\mu}\right)$
- $I_2.\,|\mathcal{E}_i| = |\mathcal{M}_i|$

Algorithm 1 presents the pseudo-code of the monitoring algorithm while Alg. 2 presents the pseudo-code for convergecast/broadcast process. The inputs to the monitoring algorithm are $\mathcal{M}_i$, $\overrightarrow{\mathcal{E}_i}$ (depending on how it is defined), Γ_i, ϵ_1 or ϵ_2, C_ω. For each problem instance I_1 and I_2, each node initializes its local vectors $\overrightarrow{\mathcal{K}_i}$, $\overrightarrow{\mathcal{A}_{i,j}}$ and $\overrightarrow{\mathcal{H}_{i,j}}$. The algorithm is entirely event-driven. Events can be one of the following: (i) change in local data $\mathcal{M}_i$, (ii) receipt of a message, and (iii) change in Γ_i. In any of these cases, the node checks if the condition of the theorem holds. Based on the value of its knowledge $\overrightarrow{\mathcal{K}_i}$, the node selects the active region $R \in C_\omega$ such that $\overrightarrow{\mathcal{K}_i} \in R$. If no such region exists, $R = \emptyset$. If $R = \emptyset$, then $\overrightarrow{\mathcal{K}_i}$ lies in the tie region and hence P_i has to send all its data. On the other hand, if $R \neq \emptyset$ the node can rely on the result of Theorem 5.1 to decide whether to send a message. If for all $P_j \in \Gamma_i$, both $\overrightarrow{\mathcal{A}_{i,j}} \in R$ and $\overrightarrow{\mathcal{H}_{i,j}} \in R$, P_i does nothing; else it needs to set $\overrightarrow{\mathcal{S}_{i,j}}$ and $|\mathcal{S}_{i,j}|$. Based on the conditions of the Theorem, note that these are the only two cases when a node needs to send a message. Whenever it receives a message ($\overrightarrow{\mathcal{S}}$ and $|\mathcal{S}|$), it sets $\overrightarrow{\mathcal{S}_{j,i}} \leftarrow \overrightarrow{\mathcal{S}}$ and $|\mathcal{S}_{j,i}| \leftarrow |\mathcal{S}|$. This may trigger another round of communication since its $\overrightarrow{\mathcal{K}_i}$ 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. Note that this mechanism does not enforce synchronization or affect correctness; at most it might delay convergence. This technique has been used elsewhere also [38][7].

Algorithm 1: Monitoring eigeivector/eigenvalues.

Input: ϵ_1, C_ω, $\overrightarrow{\mathcal{E}_i}$, Γ_i and L
Output: 0 if $\left|\left|\overrightarrow{\mathcal{K}_i}\right|\right| < \epsilon$, 1 otherwise
Initialization: Initialize vectors;
if MessageRecvdFrom $\left(P_j, \overrightarrow{\mathcal{S}}, |\mathcal{S}|\right)$ **then**
  $\overrightarrow{\mathcal{S}_{j,i}} \leftarrow \overrightarrow{\mathcal{S}}$;
  $|\mathcal{S}_{j,i}| \leftarrow |\mathcal{S}|$;
  Update vectors;
if $\mathcal{M}_i$, Γ_i *or* $\mathcal{K}_i$ *changes* **then**
  forall the $P_j \in \Gamma_i$ **do**
    if $LastMsgSent > L$ *time units ago* **then**
      if $R = \emptyset$ **then**
        $\overrightarrow{\mathcal{S}_{i,j}} \leftarrow \frac{|\mathcal{K}_i|\overrightarrow{\mathcal{K}_i} - |\mathcal{S}_{j,i}|\overrightarrow{\mathcal{S}_{j,i}}}{|\mathcal{K}_i| - |\mathcal{S}_{j,i}|}$;
        $|\mathcal{S}_{i,j}| \leftarrow |\mathcal{K}_i| - |\mathcal{S}_{j,i}|$;
      if $\overrightarrow{\mathcal{A}_{i,j}} \notin R$ *or* $\overrightarrow{\mathcal{H}_{i,j}} \notin R_\ell$ **then**
        Set $\overrightarrow{\mathcal{S}_{i,j}}$ and $|\mathcal{S}_{i,j}|$ such that $\overrightarrow{\mathcal{A}_{i,j}}$ and $\overrightarrow{\mathcal{H}_{i,j}} \in R_\ell$;
      MessageSentTo$\left(P_j, \overrightarrow{\mathcal{S}_{i,j}}, |\mathcal{S}_{i,j}|\right)$;
      $LastMsgSent \leftarrow$ CurrentTime;
      Update all vectors;
    else Wait L time units and then check again;

The monitoring algorithm raises a flag whenever either $\left\|I_1.\overrightarrow{\mathcal{K}_i}\right\| > \epsilon$ or $\left\|I_2.\overrightarrow{\mathcal{K}_i}\right\| > \epsilon$. Once the flag is set to 1, the nodes engage in a convergecast-broadcast process to accumulate data up the root of the tree, recompute the model and disseminate it in the network.

For the mean monitoring algorithm in the convergecast phase, whenever a flag is raised, each leaf node in the tree forwards its local mean up the root of the tree. In this phase, each node maintains a user selected alert mitigation constant, τ which ensures that an alert is stable for a given period of time τ for it to send the data. Experimental results show that this is crucial in preventing a false alarm from progressing, thereby saving resources. In order to implement this, whenever the monitoring algorithm raises a flag, the node notes the time, and sets a timer to τ time units. Now, 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. If it has been recorded τ or more time units ago, the node does one of the following. If it has received messages from all its neighbors, it recomputes the

new mean, sends it to all its neighbors and restarts its monitoring algorithm with the new mean. On the other hand, if it has received the mean from all but one of the neighbors, it combines its data with all of its neighbors' data and then sends it to the neighbor from which it has not received any data. Other than any of these cases, a node does nothing.

For the eigenvector monitoring, in place of sending a local mean vector, each node forwards the covariance matrix $\mathcal{C}_i$. Any intermediate node accumulates the covariance matrix of its children, adds it local matrix and sends it to its parent up the tree. The root computes the new eigenvectors and eigenvalues. The first eigenstate is passed to the monitoring algorithm.

F. Correctness and complexity analysis

The eigen monitoring algorithm is eventually correct.

Theorem 5.2: The eigen monitoring algorithm is **eventually correct**.

Proof: For the eigen monitoring algorithm, the computation will continue for each node unless one of the following happens:

- for every node, $\overrightarrow{\mathcal{K}_i} = \overrightarrow{\mathcal{E}^G}$
- for every P_i and every neighbor P_j, $\overrightarrow{\mathcal{K}_i}$, $\overrightarrow{\mathcal{A}_{i,j}}$,and $\overrightarrow{\mathcal{H}_{i,j}}$ are in the same convex region $R \in C_\omega$.

In the former case, every node obviously computes the correct output since the knowledge of each node becomes equal to the global knowledge. In the latter case, Theorem 5.1 dictates that $\overrightarrow{\mathcal{E}^G} \in R$. Note that by construction, the output of the thresholding function (in this case $\|\overrightarrow{x}\| > \epsilon$) is invariant inside any $R \in C_\omega$. In other words, the binary function $\left\|\overrightarrow{\mathcal{E}^G}\right\| < \epsilon$ and $\left\|\overrightarrow{\mathcal{K}_i}\right\| < \epsilon$ will have the same output inside R. Therefore in either of the cases, the eigen monitoring algorithm is correct.

Moreover, since $\mathcal{C} = \frac{1}{\sum_{j=1}^{n} |\mathcal{M}_i|} \sum_{j=1}^{n} \mathcal{C}_i$ (see Eqn. 1) and $\overrightarrow{\mu} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{|\mathcal{M}_i|} \overrightarrow{x_{i,j}}}{\sum_{j=1}^{|\mathcal{M}_i|} |\mathcal{M}_i|}$ the models built are also the same compared to a centralized algorithm having access to all the data. ■

Determining the communication complexity of local algorithms in dynamic environments is still an open research issue. Researches have proposed definitions of locality [7][38]. Note that for an exact algorithm as the eigen monitoring algorithm, the worst case communication complexity is O(size of network). This can happen, for example, when the each node has a vector in a different convex region and the global average is in another different region. However, as shown in this paper and also by several authors [38][7] there are several problem instances for which the resource consumption becomes independent of the size of the network. Interested readers

Algorithm 2: P2P Eigen-monitoring Algorithm.

Input: ϵ_1, ϵ_2, C_ω, $\mathcal{M}_i$, Γ_i, L, τ

Output: (i) $\overrightarrow{V}, \Theta$ such that $\left\|\mathcal{C} \cdot \overrightarrow{V} - \Theta \cdot \overrightarrow{V}\right\| < \epsilon_1$ and $||\overrightarrow{V}|| = 1$, (ii) $\overrightarrow{\mu}$ such that $\left\|\frac{\sum_{i=1}^{n}\sum_{j=1}^{|\mathcal{M}_i|}\overrightarrow{x_{i,j}}}{\sum_{j=1}^{|\mathcal{M}_i|}|\mathcal{M}_i|} - \overrightarrow{\mu}\right\| < \epsilon_2$

Initialization

begin

- Initialize vectors;
- *MsgType* = MessageRecvdFrom(P_j);

if $MsgType = Monitoring\_Msg$ **then** Pass Message to appropriate Monitoring Algorithm;

if $MsgType = New\_Model\_Msg$ **then**

- Update $\overrightarrow{V}$, Θ, $\overrightarrow{\mu}$;
- Forward new model to all neighbors;
- Datasent=false;
- Restart Monitoring Algorithm with new models;

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 τ time **then**
      - D_1=$\mathcal{C}_i$+ $Recvd\_Covariance$;
      - D_2=$\frac{\sum_{j=1}^{|\mathcal{M}_i|}\overrightarrow{x_{i,j}}}{|\mathcal{M}_i|} + Recvd\_Mean$;
      - $Datasent$=true;
      - Send D_1, D_2 to remaining neighbor
    - **else** *DataAlert*=CurrentTime;
- **if** Received from all neighbors **then**
  - D_1=$\mathcal{C}_i$+ $Recvd\_Covariance$;
  - D_2=$\frac{\sum_{j=1}^{|\mathcal{M}_i|}\overrightarrow{x_{i,j}}}{|\mathcal{M}_i|}$+ $Recvd\_Mean$;
  - $(\overrightarrow{V}, \Theta)$=EigAnalysis($D_1$) where $||\overrightarrow{V}|| = 1$;
  - $\overrightarrow{\mu}$ =Mean(D_2);
  - Forward new $\overrightarrow{V}, \Theta, \overrightarrow{\mu}$ to all neighbors;
  - *Datasent*=false;
  - Restart Monitoring Algorithm with new models;

if $\mathcal{M}_i$, Γ_i *or* $\overrightarrow{\mathcal{K}_i}$ *changes* **then**

- Run Monitoring Algorithm;
- *flag*=Output_Monitoring_Algorithm();
- **if** flag=*1 and* P_j=*IsLeaf()* **then**
  - Execute the same conditions as $MsgType = Dataset\_Msg$

are referred to [6] for a detailed discussion on communication complexity and locality of such algorithms.

VI. Experimental results

In this section we demonstrate the experimental results of the distributed eigen monitoring algorithm. Before doing that, we describe centralized experiments showing how the fundamental plane changes with variations in local galactic density. Then we describe distributed experiments showing the performance of the eigen monitoring algorithm for a distributed streaming scenario when the same data is streamed at multiple nodes. Our goal is to demonstrate that, using our distributed eigen monitoring algorithm to compute the principal components and monitor them in a streaming scenario, we can find very similar results as were obtained by applying a centralized PCA. As an interesting aside, even though our goal was not to make a new discovery in astronomy, the results are astronomically noteworthy. We argue that our distributed algorithm could have found very similar results to the centralized approach at a fraction of the communication cost. Also, we want to emphasize that this distributed eigen monitoring algorithm can be applied to a number of change-detection applications in high-throughput streaming scenarios (such as the LSST) for important astronomical discoveries of many types. The importance and novelty of this algorithm compared to existing distributed PCA algorithms is that this is an exact algorithm that deterministically converges to the correct result.

A. Fundamental Plane results

As noted in Section IV-A, we divide the entire dataset into 30 bins. The bins are arranged from low to high density. In this section we present the results of our fundamental plane experiments for those 30 bins. We have only used the elliptical galaxies in our experiments from the SDSS and 2MASS dataset.

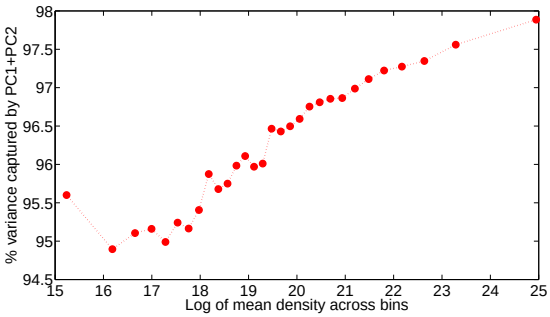
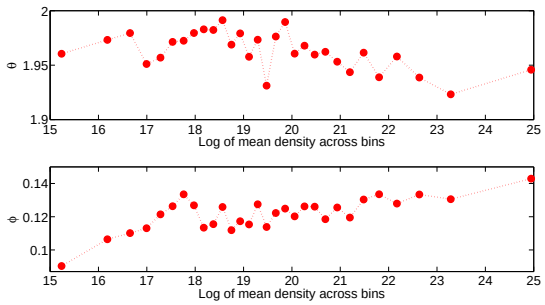
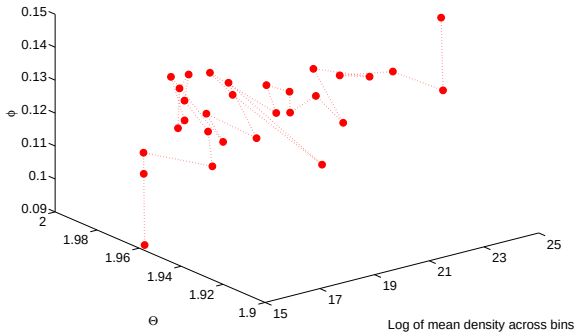


Fig. 3. Variance captured by PCs 1 and 2 w.r.t. log of mean density of each bin. Bin 1 has the lowest mean density and Bin 30 the highest.



(a) Variation of θ and ϕ independently w.r.t. log of bin density for 30 bins



(b) Variation of θ and ϕ w.r.t. log of bin density for 30 bins

Fig. 4. Plot of variation of θ and ϕ independently with bin number. The bins are numbered in increasing order of density.

Figure 3 provides the most significant scientific result. It demonstrates the dependence of the variance captured by the first 2 PC's with respect to log of bin density (the x-axis shows mean density of each bin in log-scale). As seen, the variance increases monotonically from almost 95% to 98% with increase in galactic bin density. This clearly demonstrates a new astrophysical effect, beyond that traditionally reported in the astronomical literature. This results from the application of distributed data mining (DDM) on a significantly (by 1000 times) larger set of data. More such remarkable discoveries can be anticipated when DDM algorithms of the type reported here are applied to massive scientific (and non-scientific) data streams of the future.

To analyze more deeply the nature of the variation of the first two PCs with respect to increasing galactic density, we plot the direction of the normal to the plane defined by the first 2 PCs *i.e.* pc1 and pc2. Since each of these PC's are vectors in 3-d, so is the normal to the plane. The normal vector is represented by its two directional angles: the spherical polar angles

θ and ϕ. Figure 4 shows a plot of θ and ϕ for 30 bins. Figure 4(a) shows the variation of θ and ϕ independently with $\log$ of mean galactic density. Figure 4(b) shows the variation of both with $\log$ of mean density. The systematic trend in the change of direction of the normal vector seen in Figure 4(b) is a new astronomy result. This represents exactly the type of change detection from eigen monitoring that will need to be applied to massive scientific data streams, including large astronomy applications (LSST) and large-scale geo-distributed sensor networks, in order to facilitate knowledge discovery from these petascale data collections.

B. Results of distributed PCA algorithm

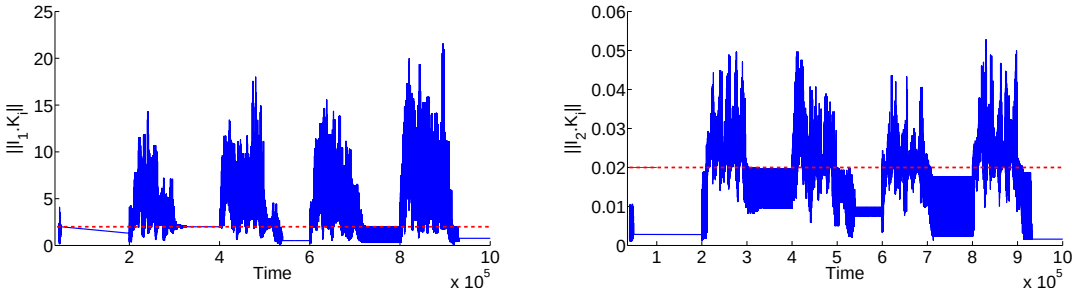
The distributed PCA implementation makes use of a Java-based simulated environment for simulating thousands of peers on a single computer. For generating realistic topologies the simulator uses BRITE [12], which is a universal topology generator from Boston University. In our simulations we used topologies generated according to the *Barabasi Albert (BA)* model. On top of the network generated by BRITE, we overlayed a spanning tree. We have experimented with varying network sizes ranging from 50 to 1000 nodes. We report all times in terms of simulator ticks since wall time is meaningless when simulating thousands of nodes on a single PC. We set up the simulator such that an edge delay of x msecs in BRITE topology corresponds to x simulator ticks. We make the assumption that the time required for local processing is trivial compared to the overall network latency and therefore, convergence time for the distributed PCA algorithm is reported in terms of the average edge delay.

We have divided the data of the centralized experiments into 5 bins (instead of 30), sorted by galactic density. Each bin represents the data distribution at a certain time in the streaming scenario and the distribution changes every 200,000 simulation ticks which we call an *epoch*. This implies that every 200,000 simulation ticks we supply the nodes with a new bin of data. The whole experiment therefore executes for 200,000$\times$5=1,000,000 simulator ticks. Furthermore, within each epoch, we stream the data at a rate of 10% of the bin size for every 10,000 simulation ticks which we call the *sub-epoch* interval. Thus, starting from the beginning of any epoch, the whole data is changed by 100,000 ticks and no data is changed for the later 100,000 ticks of that epoch. In other words, all 10,000 points are received simultaneously by all nodes at the first tick of each sub-epoch (except during the last 100,000 ticks of each epoch).

The two quantities measured in our experiments are the *quality* of the result and the *cost* of

the algorithm. For the eigen monitoring algorithm, quality can be measured as (1) the number of peers which report an agreement between the model at each node and the data *i.e.* $\left\|I_1.\overrightarrow{\mathcal{K}_i}\right\| < \epsilon_1$ or $\left\|I_2.\overrightarrow{\mathcal{K}_i}\right\| < \epsilon_2$ for each time instance, and (2) the average L2 norm distance between the principal eigen vector and the and the computed eigen vector in the distributed scenario over all the bins. For cost we measure the number of monitoring messages and the number of computation messages separately.

We have used the following default values for the algorithm: size of leaky bucket $L = 500$, error threshold $\epsilon_1 = 2.0$ $\epsilon_2 = 0.02$, alert mitigation constant $\tau = 500$, and number of peers = 50.



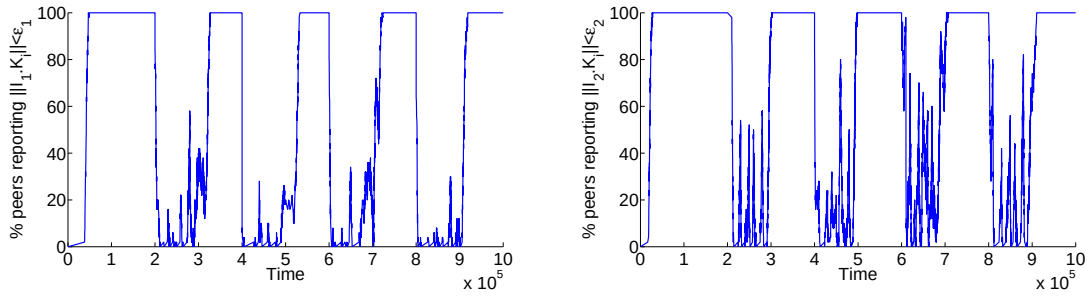
(a) $\left\|I_1.\vec{\mathcal{K}_i}\right\|$ vs. Time for eigenmonitoring (b) $\left\|I_2.\vec{\mathcal{K}_i}\right\|$ vs. Time for mean monitoring

Fig. 5. Variation of $\left\|I_1.\vec{\mathcal{K}_i}\right\|$ (left) and $\left\|I_2.\vec{\mathcal{K}_i}\right\|$ (right) across all the peers vs. Time.

Figure 5 shows the variation of the local knowledges of each peer throughout the execution of the experiment and the thresholds (red dotted lines). The left figure shows $\left\|I_1.\overrightarrow{\mathcal{K}_i}\right\|$ (eigenmonitoring) while the right figure shows $\left\|I_2.\overrightarrow{\mathcal{K}_i}\right\|$ (mean monitoring). For both the figures, the norm of the knowledge vectors exceed the respective thresholds at the beginning of each epoch (200,000, 400,000, 600,000, and 800,000 ticks), because the data corresponds to a new bin. The peers then jointly infer this disagreement using the monitoring algorithm and the convergecast/broadcast round is invoked which rebuilds and distributes a new set of eigenvectors and eigenvalues. As a result, the norm of the local knowledge at each peer drops below the corresponding threshold and only the monitoring algorithm operates for the rest of this epoch.

Accuracy and convergence of the distributed eigen monitoring algorithm is shown in Figure 6. The left figure shows the accuracy of eigen monitoring while the right one shows the same for mean monitoring. As shows, accuracy is low for the first 100,000 ticks of each epoch since the data is changing during that time. Accuracy increases to 100% during the later 100,000 ticks

since the model is in accordance with the data. This pattern is repeated for all the epochs. The convergence rate of the algorithm is shown in Figure 7 by zooming in to the second epoch. The data is changed at every 10,000 ticks between 200,000 and 300,000 ticks. This is why the accuracy is low during this period. The algorithm converges to 100% accuracy within 330,000 ticks *i.e.* within 30,000 ticks after the data stops changing. The average edge delay is 1000 simulator ticks Hence the algorithm converges in approximately 30 times the average edge delay. Figure 8 shows the messages exchanged per peer throughout the experiment. The monitoring messages, shown in the left figure, increase whenever the data changes but decreases once the algorithm converges. The number of messages exchanged during the stationary period is very low compared to an algorithm which broadcasts all the information every sub-epoch. The rate of messages of the latter is 2 per sub-epoch (considering two neighbors per peer on average). The data messages is shown as cumulative plot in the right figure. As shown there is an high number of data messages for each epoch change and it decreases for the later 100,000 of all epochs. For any experiment, new models are build 2 to 3 times per epoch.



(a) Accuracy vs. Time for eigen monitoring

(b) Accuracy vs. Time for mean monitoring

Fig. 6. Percentage of peers agreeing to $\left\|I_1.\vec{\mathcal{K}_i}\right\| < \epsilon_1$ (left figure) and $\left\|I_2.\vec{\mathcal{K}_i}\right\| < \epsilon_2$ (right figure). As clearly shown, the algorithm shows high accuracy.

The last set of experiments show that the quality of the models built by the algorithm and its communication complexity is independent of the number of nodes in the network, thereby guaranteeing high scalability. We first compare the quality of the models build by the distributed eigen monitoring algorithm to that of a centralized algorithm having access to all the data. Since we compute the principal eigen vector for each bin separately, we plot the average L2 norm distance between the centralized and distributed eigen vectors for every experiment. The experiments have been repeated for 10 independent trials. Figure 9 shows the quality of

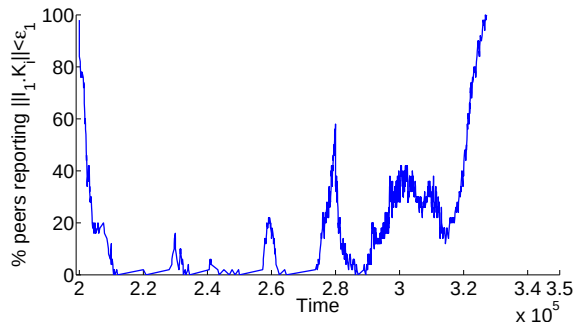
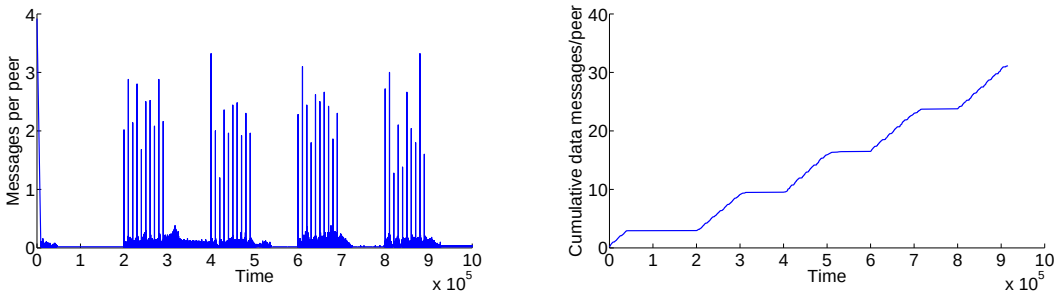


Fig. 7. Convergence of the monitoring algorithm to 100% accuracy.



(a) Monitoring messages vs. Time (b) Cumulative data messages vs. Time

Fig. 8. Messages exchanged by the eigen monitoring algorithm per peer throughout the experiment.

the computed models for different network sizes. As shown in the figure, the proposed eigen monitoring algorithm produces results which are highly accurate compared to their centralized counterpart. Moreover, quality does not degrade with increasing network sizes. Because our algorithm is provably correct, the number of nodes has no influence on the quality of the result.

Figures 10 and 11 show the number of messages exchanged per node when the number of nodes is increased from 50 to 1000. In this context, normalized message per node means the number of messages sent by a node per unit of sub-epoch (*i.e.* every data change). This is the maximal rate at which any node can send messages in our distributed algorithm. Since the length of each sub-epoch is 10,000 ticks and L=500, this maximal rate is therefore, 10,000/500$\times$2=40, assuming two neighbors per node, on average. Also, for an algorithm which uses broadcast as the communication model, its normalized messages will be 2 per sub-epoch assuming two neighbors per node, on average. In all our experiments, the normalized messages per peer is close to 0.3, well below these maximal rates. Thus the proposed algorithm is highly efficient with respect to

communication. Also as shown, the monitoring messages remain constant even if the number of nodes is increased. This demonstrates excellent scalability of the algorithm.

Finally, we also plot the number of times data is collected per epoch. In most cases, the number of such convergecast rounds is 3 per epoch. Note that this can be reduced further by using a larger alert mitigation constant τ, or larger error threshold ϵ_1 or ϵ_2.

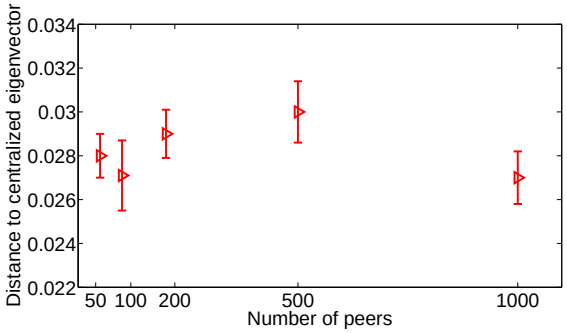


Fig. 9. L2 norm distance between distributed and centralized eigenvectors vs. number of nodes. This remains the same thereby showing good accuracy. Plotted are average and standard deviation over multiple trials.

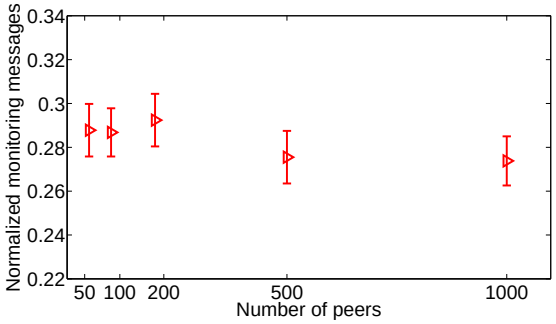


Fig. 10. L2 messages vs. number of nodes. Number of messages remain constant showing excellent scalability.

VII. PADMINI–A **P**EER-TO-PEER **A**STRONOMY **D**ATA **MINI**NG SYSTEM

PADMINI is a web based Peer to Peer data mining system that aims at being a computation tool for the researchers and users related to the field of astronomy and data mining. There are several challenges to centralizing the massive astronomy catalogs (some of which has been elucidated in the previous section) and running traditional data mining algorithms. To solve this data avalanche, PADMINI is powered by a back end peer to peer computation network to provide

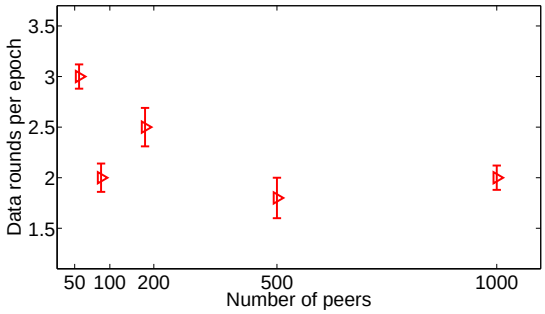


Fig. 11. Number of convergecast rounds per epoch vs. number of nodes. In most cases the convergecast round is less than 3 per epoch.

the required scalability. The back end computation network supports two distributed computation frameworks, namely Distributed Data Mining Toolkit (DDMT) [17] and Hadoop [24]. The web based PADMINI system is available online at `http://padmini.cs.umbc.edu/`. In the next few sections we first describe the different components of the system and then describe the implementation details.

A. System components

The system architecture is shown in Figure 1. It consists of a web server, DDM server, server database, jobs database and the back-end P2P network. Each of the components are discussed in details next.

1) Web server: The web server hosts the main website and is the primary interface for submitting jobs and retrieving results of the submitted jobs. Each new user signs up for an account on the website and sets up a job to be run on the system. Every user has a dedicated profile page where the user can keep a track of the jobs submitted by him. The current status of the jobs and a projected time for the completion of the jobs are also displayed on the same page. Each job submitted by the user will trigger a distributed algorithm to run on the back-end P2P network. The results of the algorithm will be pushed back to the web server. The user can then download a copy of the results of their jobs. The web service methods exposed by the DDM server are used by the web server to start a job and receive results. The web server is thus the consumer of the web service methods exposed by the DDM server.

2) Server database: The server database primarily deals with user and identity management. The database stores the information related to the registered users of the system and the privileges

they have. The job activity details of an user are also stored in this database. These include the inputs submitted by the user, the algorithm selected, the output of the job etc. The list of the supported astronomy data catalogs and their attributes that a user can use as inputs are also stored in this database.

3) DDM server: The Distributed Data Mining (DDM) Server is an intermediate tier between the web server and the back-end peer to peer computation network. The multiple job requests coming in from the web server are directed to the DDM Server and stored in a job queue. The jobs are then submitted serially for completion to the back-end computation network. The DDM server exposes a set of methods that can be used to set up jobs and for pushing the results of the completed jobs onto the web server. The web service methods encourage openness. Hence, a new system can be easily built around the available back-end P2P computation network.

4) Jobs database: The jobs database persists the book-keeping information related to the jobs. This includes the list of all the jobs that are submitted by the user, including the ones not yet submitted to the computation network. The status of the running and the waiting jobs and the results of the recently completed jobs is stored here. The database also stores information related to the back-end P2P computation network. This includes the information pertaining to the total number of active nodes, failed nodes etc.

5) P2P network: The peer to peer network forms the backbone of the back-end computation framework. All the peers in this network are configured to support two computation frameworks, namely Distributed Data Mining Toolkit (DDMT) and Hadoop. The type of jobs the user can submit is restricted by the algorithms supported by the system. Some algorithms are implemented using the DDMT while some are built on top of the Hadoop framework. The DDM server picks up a job from the queue and assigns it to be executed on top of the appropriate framework. This information depends on the type of the job and hence is implicitly set by the user.

B. Implementation details

1) Language: The website is developed using HTML, Javascript and JSPs. DDMT is implemented in Java and is based on Java Agent Development (JADE) Framework. The important methods like starting a job, stopping it, providing input etc. have been exposed as web service methods. This enables future systems to be built around the existing computation network. Hadoop provides an extensive Java API using which highly scalable Map Reduce algorithms

can be created. For running either DDMT or Hadoop, Java support is the only expected feature from a peer. Thus, the P2P computation network can be easily expanded.

2) Databases: MySQL is used as the database in the web server database as well as in the jobs database. Hibernate is used for object-relational mapping at the web server database end. Classes corresponding to the database tables make sure that operations made on the class objects get reflected and persisted in the database. Such a system not only saves development time, but also guarantees a robust database system.

3) Web service: Axis2 is used as the core engine for web services. Axis2 is built on a new architecture that was designed to be a much more flexible, efficient, and configurable. With the new Object Model defined by Axis2, it is easier to handle SOAP messages. All the web service requests are directed to the DDM Server. The DDM Server then calls the corresponding methods and starts the requested job. Axis2 also has excellent support for sending binary data or files using SOAP messages. This eases moving the inputs and outputs between the web server and the DDM server.

4) User interface: An user needs to sign up on the home page to get an account and start submitting jobs. On signing up, each user gets a personal profile page. Each algorithm supported by the website has a dedicated page on which the user can create and submit a specific job. The user can then track the status of the submitted jobs and also store the results of the most recently completed jobs on the profile page. The Google Maps interface on the PADMINI website aids an astronomer in specifying an area of the sky intuitively and effectively. The controls to select the astronomy catalogs and the supported attributes are also provided. Thus, a job can be specified with only a few clicks and the user does not to need to wait for the results.

VIII. Conclusion

This paper presents a local and completely asynchronous algorithm for monitoring the eigenstates of distributed and streaming data. The algorithm is efficient and exact in the sense that once computation terminates, each node in the network computes the globally correct model. We have taken a relatively well understood problem in astronomy — that of galactic fundamental plane computation and shown how our distributed algorithm can be used to arrive at the same results without any data centralization. We argue that this might become extremely useful when petabyte scale data repositories such as the LSST project start to generate high throughput data

streams which need to be co-analyzed with other data repositories located at diverse geographic location. For such large scale tasks, distributing the data and running the algorithm on a number of nodes might prove to be cost effective. Our algorithm is a first step to achieving this goal. Experiments on current SDSS and 2MASS dataset show that the proposed algorithm is efficient, accurate, and highly scalable.

ACKNOWLEDGMENTS

This research is supported by the NASA Grant NNX07AV70G and the AFOSR MURI Grant 2008-11. K. Das completed the research for this paper at University of Maryland, Baltimore County. C. Giannella completed the research for this paper while being an Assistant Professor of Computer Science at Loyola College in Maryland and New Mexico State University. The paper was approved for Public Release, unlimited distribution, by The MITRE Corporation: 10-0814; c2010-The MITRE Corporation, all rights reserved. The authors would also like to thank Sugandha Arora and Wesley Griffin for helping with the experimental setup.

REFERENCES

[1] H. Ang, V. Gopalkrishnan, S. Hoi, and W. Ng. Cascade RSVM in Peer-to-Peer Networks. In *Proceedings of PKDD'08*, pages 55–70, 2008.

[2] The AUTON Project. `http://www.autonlab.org/autonweb/showProject/3/`.

[3] W. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive Distributed Top-K Retrieval in Peer-to-Peer Networks. In *Proceedings of ICDE'05*, pages 174–185, 2005.

[4] N. M. Ball and R. J. Brunner. Data Mining and Machine Learning in Astronomy. arXiv:0906.2173v1, 2009.

[5] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The Price of Validity in Dynamic Networks. *Journal of Computer and System Sciences*, 73(3):245–264, 2007.

[6] 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.

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

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

[9] K. Borne. Scientific Data Mining in Astronomy. In *Next Generation of Data Mining*, chapter 5, pages 91–114. CRC press, 2009.

[10] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip Algorithms: Design, Analysis, and Applications. In *IEEE Infocom*, volume 3, pages 1653–1664, 2005.

[11] J. Branch, B. Szymanski, C. Giannella, R. Wolff, and H. Kargupta. In-Network Outlier Detection in Wireless Sensor Networks. In *Proceedings of ICDCS'06*, page 51, 2006.

[12] Boston University Representative Internet Topology Generator. http://www.cs.bu.edu/brite/.

[13] The ClassX Project: Classifying the High-Energy Universe. http://heasarc.gsfc.nasa.gov/classx/.

[14] DAME: DAta Mining and Exploration. http://voneural.na.infn.it/.

[15] S. Datta, C. Giannella, and H. Kargupta. Approximate Distributed K-Means Clustering over a Peer-to-Peer Network. *IEEE Transactions on Knowledge and Data Engineering*, 21(10):1372–1388, 2009.

[16] Digital Dig - Data Mining in Astronomy. http://www.astrosociety.org/pubs/ezine/datamining.html.

[17] The Distributed Data Mining Toolkit. http://www.umbc.edu/ddm/wiki/software/DDMT.

[18] CGAL: Delaunay Triangulation. http://www.cgal.org/.

[19] Data Mining Grid. http://www.datamininggrid.org/.

[20] H. Dutta, C. Giannella, K. Borne, and H. Kargupta. Distributed Top-K Outlier Detection from Astronomy Catalogs using the DEMAC System. In *Proceedings of SDM'07*, 2007.

[21] Elliptical Galaxies: Merger Simulations and the Fundamental Plane. http://irs.ub.rug.nl/ppn/244277443.

[22] Framework for Mining and Analysis of Space Science Data. http://www.itsc.uah.edu/f-mass/.

[23] GRIST: Grid Data Mining for Astronomy. http://grist.caltech.edu.

[24] Hadoop Home Page. http://hadoop.apache.org/.

[25] T. Hinke and J. Novotny. Data Mining on NASA's Information Power Grid. In *Proceedings of HPDC'00*, page 292, 2000.

[26] L. Huang, X. Nguyen, M. Garofalakis, M. Jordan, A. Joseph, and N. Taft. Distributed PCA and Network Anomaly Detection. Technical Report UCB/EECS-2006-99, EECS Department, University of California, Berkeley, 2006.

[27] International Virtual Observatory. http://www.ivoa.net.

[28] H. Kargupta and P. Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery*. MIT Press, 2000.

[29] H. Kargupta, W. Huang, K. Sivakumar, and E. L. Johnson. Distributed Clustering Using Collective Principal Component Analysis. *Knowledge and Information Systems*, 3(4):422–448, 2001.

[30] H. Kargupta and K. Sivakumar. *Existential Pleasures of Distributed Data Mining. Data Mining: Next Generation Challenges and Future Directions*. AAAI/MIT press, 2004.

[31] D. Kempe, A. Dobra, and J. Gehrke. Computing Aggregate Information Using Gossip. In *Proceedings of FOCS'03*, pages 482–491, 2003.

[32] 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.

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

[34] C. Meyer. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics (SIAM), 2001.

[35] US National Virtual Observatory. http://www.us-vo.org/.

[36] W.E. Schaap. *The Delaunay Tessellation Field Estimator*. PhD thesis, University of Groningen, 2007.

[37] I. Sharfman, A. Schuster, and D. Keren. A Geometric Approach to Monitoring Threshold Functions Over Distributed Data Streams. *ACM Transactions on Database Systems*, 32(4):23, 2007.

[38] 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*, 21(4):465–478, 2009.

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

APPENDIX

Now we show that if the bounds (2) and (3) hold, then the relaxed problem statement holds with $\epsilon = \epsilon_1 + \epsilon_2^2$. To do so, we must introduce more notation.

Let $\overrightarrow{\mu(\mathcal{G})}$ denote the column mean vector for $\mathcal{G}$, the global dataset. Let $\overrightarrow{\mu}$ denote the column mean vector computed the last time the model was rebuilt (the last convergecast) – an estimation of $\overrightarrow{\mu(\mathcal{G})}$. Let $\Delta\overrightarrow{\mu}$ denote $\overrightarrow{\mu(\mathcal{G})} - \overrightarrow{\mu}$. If bound (3) holds, then $||\Delta\overrightarrow{\mu}|| < \epsilon_2$.

Let $\mathcal{C}(\mathcal{G})$ denote the covariance matrix of $\mathcal{G}$, the global dataset. Let $\mathcal{C}$ denote the estimation of the covariance matrix generated by mean-shifting using $\overrightarrow{\mu}$. Specifically, the (i, j) entry of $\mathcal{C}$ is defined to be

$$\mathcal{C}(i,j) = \frac{\sum_{k=1}^{|\mathcal{G}|}(x_{k,i} - \mu_i)(x_{k,j} - \mu_j))}{|\mathcal{G}|}$$

where μ_i and μ_j are the i^{th} and j^{th} components of $\overrightarrow{\mu}$; $x_{k,i}$ and $x_{k,j}$ are the i^{th} and j^{th} components of the k^{th} data vector in $\mathcal{G}$. Let $\overrightarrow{V}$ and θ denote a vector and number computed the last time the model was built such that $||\overrightarrow{V}|| = 1$ and which satisfies bound (2), $||\mathcal{C}\overrightarrow{V} - \theta\overrightarrow{V}|| < \epsilon_1$.
Now we can state precisely the statement we will prove: if $||\mathcal{C}\overrightarrow{V} - \theta\overrightarrow{V}|| < \epsilon_1$ (bound (2)) and $||\overrightarrow{V}|| = 1$ and $||\Delta\overrightarrow{\mu}|| < \epsilon_2$ (bound (3)), then $||\mathcal{C}(\mathcal{G})\overrightarrow{V} - \overrightarrow{V}\theta|| \leq \epsilon_1 + \epsilon_2^2$. The proof proceeds as follows.
Straight-forward algebraic manipulations show that $\mathcal{C}(\mathcal{G})$ is a rank-one update of $\mathcal{C}$.

$$\mathcal{C}(\mathcal{G}) = \mathcal{C} + \Delta\overrightarrow{\mu}(\Delta\overrightarrow{\mu})^T \tag{4}$$

Thus, with $||.||_F$ denoting the Frobenius norm and $Tr(.)$ the matrix trace, $||\mathcal{C}(\mathcal{G})\overrightarrow{V} - \overrightarrow{V}\theta||$ equals

$$
\begin{aligned}
||\mathcal{C}(\mathcal{G})\overrightarrow{V} - \overrightarrow{V}\theta|| &= |||[\mathcal{C}\overrightarrow{V} - \overrightarrow{V}\theta] + [-\Delta\overrightarrow{\mu}(\Delta\overrightarrow{\mu})^T\overrightarrow{V}]||| && \text{[By (4)]} \\
&\leq \epsilon_1 + ||\Delta\overrightarrow{\mu}(\Delta\overrightarrow{\mu})^T\overrightarrow{V}]|| && \text{[By the triangle inequality and bound (2)]} \\
&\leq \epsilon_1 + ||\Delta\overrightarrow{\mu}(\Delta\overrightarrow{\mu})^T||_F||\overrightarrow{V}|| && \text{[By (5.2.2) in [34]]} \\
&= \epsilon_1 + \sqrt{Tr[(\Delta\overrightarrow{\mu}(\Delta\overrightarrow{\mu})^T)^2]} && \text{[By (5.2.1) in [34] and } ||\overrightarrow{V}|| = 1\text{]} \\
&= \epsilon_1 + ||\Delta\overrightarrow{\mu}||^2 && \text{[By straight-forward algebraic manipulations]} \\
&\leq \epsilon_1 + \epsilon_2^2 && \text{[By bound (3)]}
\end{aligned}
$$