Communication-efficient Algorithms for Distributed Stream Mining

Moshe Gabel
Communication-efficient Algorithms for Distributed Stream Mining

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Moshe Gabel

Submitted to the Senate of the Technion — Israel Institute of Technology
Tammuz 5777 Haifa July 2017
This research was carried out under the supervision of Prof. Assaf Schuster in the Faculty of Computer Science, and Prof. Daniel Keren in the University of Haifa.

Some results in this thesis have been published as articles by the author and research collaborators in conferences and journals during the course of the author’s doctoral research period, the most up-to-date versions of which being:

<table>
<thead>
<tr>
<th>Authors</th>
<th>Title</th>
<th>Conference/Proceedings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moshe Gabel, Daniel Keren, and Assaf Schuster</td>
<td>Communication-efficient distributed variance monitoring and outlier detection for multivariate time series.</td>
<td>28th IEEE International Parallel &amp; Distributed Processing Symposium (IPDPS), 2014.</td>
</tr>
<tr>
<td>Moshe Gabel, Daniel Keren, and Assaf Schuster</td>
<td>Monitoring least squares models of distributed streams.</td>
<td>Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2015.</td>
</tr>
</tbody>
</table>

**Acknowledgements**

To my advisors, my family, and my friends: thank you for your guidance, endless patience, and unwavering support. I could never have gotten so far without you!

The generous financial help of the Technion is gratefully acknowledged.
Contents

List of Figures

Abstract

1 Introduction

1.1 This Work

1.2 A Common Approach: Approximation via Threshold Monitoring

1.3 Related Work

2 Communication-efficient Distributed Variance Monitoring and Outlier Detection for Multivariate Time Series

2.1 Latent Fault Detection

2.2 Distributed Online Variance Monitoring

2.3 Distributed Outlier Detection

2.4 Evaluation
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.1</td>
<td>Performance</td>
<td>27</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Slack and Reference Point Prediction</td>
<td>29</td>
</tr>
<tr>
<td>3</td>
<td>Practical Entropy Approximation for Distributed Streams</td>
<td>31</td>
</tr>
<tr>
<td>3.1</td>
<td>Problem Definition and Notation</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>The CIDER Estimator</td>
<td>34</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Basic Geometric Monitoring</td>
<td>35</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Additive Slack and Drift</td>
<td>38</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Extending the Domain of $H_{\text{MLE}}$</td>
<td>39</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Entropy Sketch</td>
<td>40</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Dynamically-sized Sliding Windows</td>
<td>41</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Violation Resolution</td>
<td>42</td>
</tr>
<tr>
<td>3.2.7</td>
<td>Approximating Rényi and Tsallis Entropies</td>
<td>43</td>
</tr>
<tr>
<td>3.3</td>
<td>Empirical Evaluation</td>
<td>44</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Network Traffic Monitoring</td>
<td>44</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Load Balancing</td>
<td>47</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Air Quality Monitoring</td>
<td>48</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Lazy Synchronization</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>Monitoring Least Squares Models of Distributed Streams</td>
<td>51</td>
</tr>
<tr>
<td>4.1</td>
<td>Problem Definition</td>
<td>52</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Monitoring OLS of Distributed Streams</td>
<td>52</td>
</tr>
<tr>
<td>4.2</td>
<td>Monitoring Distributed Least Squares With Convex Subsets</td>
<td>54</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Notation</td>
<td>55</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Convex Safe Zones</td>
<td>56</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Infinite and Sliding Window</td>
<td>57</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Norm Constraint and the Sliding Window</td>
<td>59</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Regularization and Variants</td>
<td>59</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Deriving the Sliding Window Constraint</td>
<td>60</td>
</tr>
<tr>
<td>4.2.7</td>
<td>Deriving the Infinite Window Constraint</td>
<td>62</td>
</tr>
<tr>
<td>4.2.8</td>
<td>Window Size And Dimensions</td>
<td>62</td>
</tr>
<tr>
<td>4.3</td>
<td>Evaluation</td>
<td>64</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Synthetic Datasets</td>
<td>64</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Traffic Monitoring</td>
<td>68</td>
</tr>
<tr>
<td>4.3.3</td>
<td>GLS on Gas Sensor Time Series</td>
<td>69</td>
</tr>
<tr>
<td>5</td>
<td>Conclusions</td>
<td>71</td>
</tr>
</tbody>
</table>

Hebrew Abstract
List of Figures

2.1 Global, local, reference, and drift vectors with two nodes. .......................... 19
2.2 Safe zones for $L = 0.5, H = 1.5$ where $V(0) = (0.5, 1)$. .......................... 20
2.3 Sign test performance at different $k, f$ combinations, compared to centralized test. .......................................................... 28
2.4 Sign test performance with reference point prediction disabled. ............... 28
2.5 Comparison of different variants of the monitoring algorithm. .................. 29

3.1 Safe zone for upper threshold. ................................................................. 36
3.2 Global, local, drift, and slack vectors with two nodes. ............................. 38
3.3 $g(x) = -x \ln x$ and its concave “extension” to the negative domain, $\tilde{g}(x)$. 39
3.4 Entropy of source IP addresses in the CTU1 dataset. ............................... 45
3.5 Communication for CIDER at different approximation accuracies when monitoring source IP entropy in the CTU datasets. ................................. 46
3.6 Larger sliding windows when monitoring the CTU1 dataset. ....................... 46
3.7 Communication vs. number of simulated nodes $m$ in the CTU1 dataset. .... 47
3.8 Entropy of the WC98 dataset and CIDER’s communication. .................... 48
3.9 Entropy of the TAQ dataset and CIDER’s communication. ....................... 49
3.10 Communication vs. max number of balancing nodes on different datasets. 50

4.1 Model fit for the traffic dataset and comparison with model error. ............ 53
4.2 Monitoring distributed OLS models is difficult. ....................................... 54
4.3 Sliding and infinite window models. ....................................................... 57
4.4 DILSQ model error and syncs per round, compared to PER(100) error. ....... 65
4.5 Communication for DILSQ and periodic algorithm tuned to achieve same max error. ............................................................. 66
4.6 Communication vs. different parameters for the fixed and drift datasets. ....... 67
4.7 Velocity measurements from 8am–9am, and interpolated velocity at 9am. ...... 68
4.8 Communication for DILSQ and periodic algorithm on the traffic dataset at different $\epsilon$ values. .......................................................... 69
4.9 Communication for DILSQ and periodic algorithm with same max model error on gas sensors data. ..................................................... 69
Abstract

Recent years have seen an explosion in the number of connected devices, which means not only growth in velocity and volume of data, but also that data sources are increasingly spatially distributed, incurring higher communication costs. Data mining algorithms often assume that data is centralized or that communication is inexpensive: the setting is implicitly assumed to be a data center. In settings like wireless sensor networks, however, communication costs limited battery power. Moreover, most work only considers one-shot computation: computing a result once from a fixed data set. Yet data is increasingly dynamic, and many applications require current results over a recent time window.

We focus on computing approximations over aggregated distributed data streams with reduced communication. We describe three novel distributed approximations for important non-linear functions: variance, Shannon’s entropy, and least-squares regression. Using a geometric safe zone approach (also called geometric monitoring), we convert global approximation bounds on the data aggregate to local threshold constraints that each node can check independently. Our algorithms provide deterministic user-defined approximation bounds, while avoiding messages unless they are needed to maintain those bounds. Compared to the centralized solution, our algorithms reduce communication by up to two orders of magnitude on several real data sets that represent real applications, including machine health monitoring, network monitoring with netflows, traffic monitoring, and others. We also extend the formulation of geometric monitoring to variable-sized windows, which is critical to achieve low communication when window sizes are dynamic and vary across nodes.
Chapter 1

Introduction

Classic data mining algorithms often assume that data is centralized, or that processing nodes can communicate fairly easily. Much of the research into distributed data mining focuses on parallelizing operations to improve throughput. Distributed machine learning algorithms tend to focus on parallel methods (e.g., mini-batches [DGBSX12]), where the setting is often implicitly assumed to be a data center or a compute cluster: constant communication between nodes is feasible and fairly inexpensive, and the goal is to simply minimize runtime. Moreover, much existing work only considers “one-shot” computations – where a result is computed once over a fixed data set. Even online learning algorithms tend to ignore the cost of sending the data from its source to the data center in the first place.

Meanwhile, the last decade has seen an explosion in the number of connected devices and sensors, as well as in the use of data mining and machine learning for increasingly diverse applications. This means not only growth in velocity and volume of data, but also that data sources are more likely to be spatially or geographically distributed, increasing communication costs. Many applications now work with ever-changing distributed data where we are interested in maintaining the value of the computed function, such as a regression model, over a recent time window [Cor13].

Consider, for example, distributed algorithms for gradient descent, often used to learn models for regression or classification problems (reviewed in Section 1.3.5). Most techniques focus on handling large datasets, where constant communication between nodes is implicitly assumed to be inexpensive, as is the case in data center settings, and the goal is to minimize runtime. Moreover, maintaining an up-to-date model as data changes requires additional communication and computations. Existing approaches such as online algorithms and warm start tend to ignore the cost of sending the data from its source to the data center in the first place, making them unsuitable for wireless sensor networks and similar settings where communication is limited. Wireless sensors are often battery-powered devices with power-efficient CPUs, where the main drain on the battery is communication [GKD+13, SL13]. Thus, maintaining an up-to-date regression model using existing approaches would incur constant power drain.
The high communication costs of traditional data mining algorithms motivates communication-efficient distributed stream mining: algorithms that continuously approximate functions over the aggregate of distributed data streams, while avoiding communication.

1.1 This Work

We focus on monitoring functions over distributed data streams with reduced communication. We employ a recent general framework called geometric monitoring [KSA+14, KSSL12], designed for communication-efficient threshold monitoring. The key observation in geometric monitoring is that low-communication monitoring of even highly-complex functions is often possible by deriving constraints on the input domain, rather than the function output. These constraints are then decomposed to local constraints on input data of each stream (or node).

We derive novel algorithms for maintaining up-to-date estimates of important nonlinear functions: variance, as part of a distributed machine health monitor (Chapter 2); Shannon’s entropy (Chapter 3); and least-squares regression models (Chapter 4). Our monitoring algorithms provide guaranteed (as opposed to probabilistic) user-defined error bounds with reduced communication. We demonstrate and evaluate these algorithms on real data sets representing real applications, including outlier detection for multivariate time series (machine health monitoring), network attack detection via netflows, traffic monitoring, and others.

Communication-efficient Distributed Variance Monitoring and Outlier Detection for Multivariate Time Series

Modern scale-out services are comprised of thousands of individual machines, which must be continuously monitored for unexpected failures. One recent approach to monitoring is latent fault detection, an adaptive statistical framework for scale-out, load-balanced systems. By periodically measuring hundreds of performance metrics and looking for outlier machines, it attempts to detect subtle problems such as misconfigurations, bugs, and malfunctioning hardware, before they manifest as machine failures. Previous work on a large, real-world Web service has shown that many failures are indeed preceded by such latent faults.

Latent fault detection is an offline framework with large bandwidth and processing requirements. Each machine must send all its measurements to a centralized location, which is prohibitive in some settings and requires data-parallel processing infrastructure. In Chapter 2 we adapt the latent fault detector to provide an online, communication- and computation-reduced version. We utilize stream processing techniques to trade accuracy for communication and computation.

We first describes a novel communication-efficient online distributed variance mon-
Monitoring algorithm that provides a continuous estimate of the global variance within guaranteed approximation bounds. Using this variance monitor, we provide an online distributed outlier detection framework for non-stationary multivariate time series common in scale-out systems. The adapted framework reduces data size and central processing cost by processing the data in situ, making it usable in wider settings. Like the original framework, our adaptation admits different comparison functions, supports non-stationary data, and provides statistical guarantees on the rate of false positives. Simulations on logs from a production system show that we are able to reduce bandwidth by an order of magnitude, with below 1% error compared to the original algorithm.

**Practical Entropy Approximation for Distributed Streams**

Entropy is a fundamental property of data and a key metric in many scientific and engineering fields. Entropy estimation has been extensively studied, but almost always under the assumption that there is a single data stream, seen in its entirety by one node running the estimation algorithm. Multiple distributed data sources are becoming increasingly common, however, with applications in signal processing, computer science, medicine, physics, and more. Centralizing all data can be infeasible, for example in networks of battery or bandwidth limited sensors, so entropy estimation in distributed streams requires new, communication-efficient approaches.

Chapter 3 proposes a practical communication-efficient algorithm for continuously approximating the entropy of distributed streams, with deterministic, user-defined error bounds. Unlike previous streaming methods, it supports deletions and variable-sized time-based sliding windows, while still avoiding communication when possible. Moreover, it optionally incorporates a state-of-the-art entropy sketch, allowing for both bandwidth reduction and monitoring very high dimensional problems. Finally, it provides the approximation to all nodes, rather than to a centralized location, which is important in settings such as wireless sensor networks.

We also extend the formulation of geometric monitoring to variable-sized windows, which is critical to achieve low communication when window sizes are dynamic and vary across nodes.

Evaluation on several public datasets from real application domains shows that our adaptive algorithm can often reduce the number of messages by two orders of magnitude, compared to centralizing all data in one node.

**Monitoring Least Squares Models of Distributed Streams**

Least squares regression is widely used to understand and predict data behavior in many fields. As data evolves, regression models must be recomputed, and indeed much work has focused on quick, efficient and accurate computation of linear regression models. In distributed streaming settings, however, periodically recomputing the global model is wasteful: communicating new observations or model updates is required even when
the model is, in practice, unchanged. This is prohibitive in many settings, such as in wireless sensor networks, or when the number of nodes is very large. The alternative, monitoring prediction accuracy, is not always sufficient: in some settings, for example, we are interested in the model’s coefficients, rather than its predictions.

Chapter 4 describes the first monitoring algorithm for multivariate regression models of distributed data streams that guarantees a bounded model error. It maintains an accurate estimate using a fraction of the communication by recomputing only when the precomputed model is sufficiently far from the (hypothetical) current global model. When the global model is stable, no communication is needed.

Experiments on real and synthetic datasets show that our approach reduces communication by up to two orders of magnitude while providing an accurate estimate of the current global model in all nodes.

1.2 A Common Approach: Approximation via Threshold Monitoring

Our common approach throughout this work is to reduce the problem of communication-efficient continuous function approximation to communication-efficient continuous threshold monitoring using geometric monitoring [KSA+14, KSSL12]. What follows is an overview of geometric monitoring using safe zones; detailed explanations are provided in individual chapters.

Overview of Geometric Monitoring with Safe Zones

Let $V^j$ where $j = 1 \ldots m$ be the current local data at each of $m$ nodes, and let $V$ be their aggregate, $V = \frac{1}{m} \sum_{i=1}^{m} V^j$. Denote by $f$ the estimated function (e.g., entropy) of the global data, expressed as a function of the global average: $f(V)$. Finally, let $T$ be a threshold constraint on $f(V)$, for example $f(V) < T$. In geometric monitoring, we decompose the global constraint $f(V) < T$ on the value of $f$ to local constraint $C$ on $V^j$ in the input (domain) of $f$ such that as long as $V^j \in C$ for all nodes $j$, then $f(V) < T$ is guaranteed. We refer to $C$ as the safe zone.

Applying Geometric Monitoring

Applying geometric monitoring to any function $f$ is done in two steps. First, we express $f$ as a function of $V$, the aggregate of local vectors $V^j$. Second, given the approximation $f_0$ and the accuracy constraint, we derive a suitable safe zone $C$. Finding safe zones for variance (Chapter 2) and Shannon’s entropy (Chapter 3) is similar, since variance is convex and entropy is concave; Section 3.2.1 formalizes this result to any convex or concave function. We derive a custom safe zone for least squares regression (Chapter 4), See individual chapters for detailed notations and explanations on applying geometric monitoring to each individual function.
The Generic Estimation Algorithm

We reduce the approximation bounds for each function to one or two threshold constraints, for example \( f_0 - \epsilon \leq f(V) \leq f_0 + \epsilon \), where \( f_0 \) is current approximation. Conceptually, each of our monitoring algorithms is a realization of the following “generic” approximation algorithm:

1. Gather initial data \( V_0 \) and compute an initial estimate \( f_0 = f(V_0) \), as well as the approximation constraint.
2. Use geometric monitoring to efficiently monitor the approximation constraint: if the approximation \( f_0 \) is sufficiently accurate (more precisely, as long as all \( V_j \) are in the safe zone), no communication takes place.
3. Otherwise, synchronize: update the approximation \( f_0 \) and its constraints (for example by repeating step 1) and continue monitoring (go to step 2).

A special coordinator node coordinates the monitoring process: computing \( f_0 \) and \( C \), resolving safe zone violations, polling nodes for local data, etc.

Though conceptually simple in theory, effectively applying the generic algorithm in practice requires modifying some of the details of the algorithm. Techniques such as drift, slack, and reference point prediction help avoid needless safe zone violations in step 2. Lazy synchronization helps avoid communication increase with \( m \) by avoiding many approximation updates in step 3. These are described in detail in individual chapters.

Extensions to Geometric Monitoring

Finally, we extend the geometric monitoring framework in several ways. Section 2.2.5 introduces multiplicative slack with dynamic slack allocation. We extend geometric monitoring to dynamically-sized sliding windows (Section 3.2.5) and introduce a novel opposite slack balancing strategy for lazy synchronization. Finally, Section 4.3 introduces the periodic oracle (PER) as a performance baseline.

1.3 Related Work

1.3.1 Distributed Stream Monitoring

Early work on distributed stream monitoring focused on tailoring protocols for basic primitives by exploiting their specific properties. Babcock and Olston [BO03] monitor top-\( k \) items by tracking the differences from the current top-\( k \) and distributing slack. Bar-Or et al. [BOKSW05] build a hierarchical decision tree by deriving bounds on the attribute selection functions, keeping track of promising attributes that are clearly better than others. Keralapura et al. [KCR06] take advantage of the additive property of counts to alert if the global count crosses a specified threshold.

Recently, more generic approaches to monitoring distributed streams have received much attention. Most work dealt with the simpler cases of linear func-
tions [KCR06, KRRS08], as well as monotonic functions [MTW05]. Some papers addressed specific non-linear problems, e.g., monitoring the value of a single-variable polynomial [SR08], and analysis of eigenvalue perturbation [HNG+07]. Wolff et al. [WBK09] propose an eventually-consistent peer-to-peer algorithm based on a consensus protocol and decomposing a functions domain to convex regions. A recent survey by Cormode [Cor13] formalizes the continuous distributed monitoring model and presents several results in this setting.

The safe zone approach we apply [KSSL12, SSK08] is a generalization of the geometric monitoring approach by Sharfman et al. [SSK07b]. In [SSK07b] a geometric framework for monitoring arbitrary functions over distributed streams was proposed, and later extended and generalized [KSSL12, LSK+15]. In this framework we begin with global constraints on the value of the monitored function, and derive local constraints that nodes can check rather to avoid communication. See Lazerson et al. [LGKS17] for a recent review.

Sketching [Mut05, CG07] is a complementary general approach to stream processing. Sketching approaches reduce the size of messages (e.g., using hash functions or random projections) without losing the ability to answer queries (e.g., number of distinct elements). Conversely, geometric monitoring reduces the number of messages sent, and provide probabilistic bounds on the error.

1.3.2 Distributed Outlier Detection

The distributed outlier detection we describe in Chapter 2 differs from existing techniques in several aspects. First, our approach holistically compares entire multivariate time series, rather than isolated observations (or univariate time series). Second, static similarity thresholds and top-\(n\) outliers approaches common in distributed settings do not adapt to changes, and may incur too many false positives. Our framework adapts to the data and its statistical guarantees limit the false positive rate. Lastly, we allow non-stationary processes, automatically adapting to concept drifts.

Many distributed outlier detection schemes have been proposed in the context of wireless sensor networks (WSN). Branch et al. [BSG+06] describe a peer-to-peer top-\(n\) outlier detector for general anti-monotonic ranking functions. While saving power overall, it transmits many more data points than the centralized version, showing that minimizing power can come at the cost of increased bandwidth. Subramaniam et al. [SPP+06] use kernel density estimation models to approximate the underlying global data distribution in a hierarchy of nodes. Nodes probabilistically send a fraction of their local kernel models up the sensor hierarchy, where they are continuously combined to maintain single model. TACO [GKD+10] by Giatrakos et al. uses a similarity-preserving sketch for univariate time series, with a similarity threshold to define outliers. Sensors send sketched versions of their data to cluster heads, which produce candidate lists of outliers verified using a transparent spanning tree approach. Our setting is somewhat
different from the WSN setting. WSN schemes aim to save power, and so minimize the number of messages regardless of their size. We focus on bandwidth, minimizing the number of transmitted values. WSN work is also frequently concerned with the network topology and reliability, often limiting communication to direct peers. We are free to assume a reliable network where nodes can directly with the coordinator (or any other node).

Two approaches in particular aim to avoid communication. For univariate time series, Huang et al. [HNG+07] apply stochastic matrix perturbation theory to derive local constraints for monitoring principal component analysis of distributed data. The estimated matrix is used to find outliers. Local updates are not sent if they cannot substantially affect the current estimation. Burdakis et al. [BD12] use a geometric monitoring approach to outlier detection, by expressing common similarity measures as functions of the aggregate of global statistics. A static threshold on similarity induces local constraints, and violations are resolved between pairs. One drawback is that the number of monitored values increases quadratically with the number of nodes.

1.3.3 Variance Monitoring

For non-distributed streams, there are several approaches to maintaining an estimate of variance over a sliding window, such as the technique by Babcock et al. [BDMO03]. However, there are few variance monitoring algorithms for distributed streams. If the variance of a sample is acceptable, one can efficiently sample from distributed streams [CMYZ10, CMYZ12] and compute the variance over the sample. The most similar work is by Sharfman et al. [SSK07a], which uses geometric monitoring in a WSN setting to alert if the global variance rises above a static threshold, i.e. threshold monitoring. Our safe zone technique provides value monitoring – we estimate the current global variance within guaranteed approximation bounds. Moreover, it is designed to reduce total bandwidth, rather than the number of sent messages.

1.3.4 Entropy Estimation

Existing work on estimating the entropy of distributed streams can be roughly divided into four types.

Sketching

Sketching approaches reduce the size of messages rather than the number of messages sent, and provide probabilistic bounds on the error. For single streams, entropy sketches have been proposed by Bhuvanagiri and Ganguly [BG06] and Chakrabarti et al. [CDBM06]. Zhao et al. [ZLO+07] approximate $x \ln x$ as a linear combination $c(x^{1+\alpha} - x^{1-\alpha})$, estimated in turn using Indyk’s $L_p$ norm sketch [Ind06]; the resulting sketch is used to estimate the entropy of origin-destination network flows in a datacenter. Harvey et al. [HNO08] use frequency moment sketches to estimate the Tsallis entropy, which is
then used to approximate the Shannon entropy. Clifford and Cosma [CC13] propose a simple unbiased sketch based on random linear projections drawn from a maximally-skewed stable distribution; it requires no tuning and has near-optimal space complexity. They also provide a detailed review of entropy sketching. CIDER provides deterministic bounds, and aims to reduce the number of messages (communication-efficiency) rather than their size. It optionally incorporates a recent entropy sketch [CC13] to achieve similar reductions in message size (and memory requirements), at the cost of making the bounds probabilistic.

Insertion-only Algorithms

Arackaparambil et al. [ABC09] describe a round-based probabilistic approximation for distributed streams in the cash-register model. After each round, nodes update a coordinator if enough new items have arrived during the round, as determined by a distributed counting ($F_1$ moment) estimator; otherwise, no messages are sent. They also use an entropy sketch [HNO08] to reduce message size.

Sampling

Sampling from distributed streams is well-studied, and Cormode provides a review [CMYZ12]. Sampling requires continuous communication even if entropy does not change: some communication is incurred per window in order to maintain the sliding window [CMYZ12, Cor13]. Moreover, sampling provides a probabilistic bound, and requires a large sample to control both variance and bias [Pan03, NBdRvS04, HNO08], yet communication grows approximately linearly with sample size [Cor13].

Theoretical Bounds

Communication complexity results use adversarial approaches to prove lower bounds on the number of bits or the number of messages needed to monitor entropy. Woodruff and Zhang [WZ12] and Arackaparambil et al. [ABC09] give such lower bounds, deterministic and probabilistic, with and without deletions. Arackaparambil et al. also use an adversarial construction to show that no nontrivial savings in communication is possible [ABC09]. Indeed, CIDER does not and cannot guarantee communication reduction for all data. Yet we see that on several real datasets communication reduction is not only possible but substantial (Section 3.3). We observe that much of the existing work on distributed entropy approximation is highly theoretical, and focuses on the worst case; there appears to be little empirical evaluation on real data (with a few notable exceptions, e.g., [ZLO+07, ABBS10]).

Finally, some of the above also propose approximations for generalizations of entropy; we briefly discuss those in Section 3.2.7.
1.3.5 Monitoring Least Squares Models

Due to the ubiquity of linear regression, a great deal of research was dedicated to solving for the regression model not only in a centralized setting, but over distributed systems as well; Sayed [Say14] provides a comprehensive survey. Typically, the distributed nodes compose a graph, each holding a portion of the data, and the goal is to solve for the regression model of the aggregated data. It is well-known that the accurate solution involves calculating a matrix-vector pair from the data (denote it $A, c$), and then calculating $A^{-1}c$. Since the global matrix-vector pair can be expressed as the sum of local pairs at the nodes, a path is defined over the graph, and the global pair is obtained by traversing this path; a Hamiltonian path is desirable, in order to reduce the time required to traverse the graph [LCL14]. Spanning trees have also been applied to this end by Paskin et al. [PGM05]. Eventually, the local estimates at the nodes converge, via message passing with neighbors, to a global consensus [MG12]. Tu and Sayed [TS12] suggest that diffusion strategies outperform consensus-seeking methods.

Variants include taking advantage of the global matrix’s sparseness in order to reduce traffic [GBT+04], and gradient-based methods run either sequentially or with some degree of parallelism [MBG10, LS06, Say14, YB04]. Such techniques were also applied in online distributed learning, where the sought classifier can sometimes be expressed as the solution of a linear regression problem [ZSTvdS14].

While efficient solutions were developed for computing the linear regression model over distributed nodes, there are, to the best of our knowledge, only very few papers dealing with monitoring it – that is, imposing local conditions which imply that the global solution did not change by more than a pre-defined amount since the last time it was computed (Section 4.2). Song et al. [SWGH13] apply a heuristic where the nodes do not broadcast if the newly arriving data conforms with the current model up to some tolerance. Bhaduri and Kargupta [BK08] use distributed monitoring to monitor the prediction error (Section 4.1.1) and quadratic fit error $R^2$ [BDG11], but not the error in the model itself. Gupta et al. [GRM13] address the one-dimensional regression problem – monitoring the ratio of two aggregated variables. We address the general, high-dimensional problem.

Nearly all previous work on geometric monitoring addressed functions which are either polynomials (typically quadratic), or defined by compositions of polynomial with simple functions such as medians and quotients. To the best of our knowledge, the problem addressed in Chapter 4 – monitoring the linear regression model (as opposed to its fit error) – was never addressed over a distributed setting. Note that the monitored function contains the highly complicated operation of matrix inversion, which is not linear or convex, and which, when written explicitly, becomes intractable even for relatively low dimensions (e.g., the analytic expression for the inverse of a $20 \times 20$ matrix involves polynomials with $20!$ monomials). Therefore, a straightforward application of previous work on geometric monitoring is impossible.
Chapter 2

Communication-efficient
Distributed Variance Monitoring
and Outlier Detection for
Multivariate Time Series

For large systems comprised of hundreds of machines or more, it is unreasonable to assume that all machines are working properly and are well configured. Monitoring is essential, since unnoticed faults might accumulate and eventually cause outages. Yet, the large number of machines makes manual monitoring impractical. Instead, machines are usually monitored by collecting and analyzing hundreds of performance counters [BGF+10, Isa07] reported by various system layers, from application-specific metrics (such as database statistics) to general metrics (such as CPU utilization).

Many existing failure detectors are inflexible [GSBB12], and most require centralizing the data in some form. Rule-based failure detectors define a set of watchdogs [Isa07] that trigger an alert whenever specified counters cross predefined thresholds. More advanced, supervised methods learn system behavior models from historical logs [BGF+10, BLDSB12], but are sensitive to workload changes and system updates [ZCG+05]. Textual console log analysis in high performance computing [OAS08, XHF+09] is more flexible, but maintaining such logs may be impractical in high-volume systems where transactions are very short, time-sensitive, and rapid; textual logs would be immense – difficult to output, store and retrieve. Finally, some unsupervised approaches [KTGN10, KGN08] rely on domain insights and system knowledge, and therefore have limited applicability.

Recent approaches to the monitoring problem [GSBB12, KGN08, KDJ+12] focus on early detection and handling of performance problems, or latent faults. These are outliers – machine behaviors that could indicate a fault yet fly under the radar of monitoring systems because they are not acute enough, or were not anticipated by
maintenance engineers. In previous work [GSBB12] we provided evidence that latent faults are common, and presented a novel unsupervised outlier detection framework for latent fault detection. In experiments on a real-world production system comprised of 4500 machines, we showed that over 20% of machine and software failures were preceded by latent faults. Furthermore, we were able to detect latent faults up to 14 days in advance of actual failures with up to 70% precision and 2% false positive rate – comparable to state of the art supervised techniques in controlled settings [BLDSB12]. The latent fault detector is adaptable, requires no domain knowledge, no historical logs, and no parameter tuning in the face of workload changes and software updates. It provides guarantees on false positive rates, it is non-intrusive, and it handles very large systems.

One drawback of the outlier detector is the high communication and processing costs, prohibitive in some settings. Modern data centers are large, and so too are the resultant counter logs – too large to centralize and process in one location. Parallel processing may not always be feasible, however. Furthermore, some large systems are not confined to a single datacenter but are geographically distributed. In this work we adapt the latent fault detection framework using sketching [CG07, Mut05] and safe zones [KSSL12, KSA+14, SSK08] to reduce communication and processing requirements by an order of magnitude, while preserving the framework’s advantages.

This Chapter

We make the following contributions:

1. An online, distributed statistical outlier detector framework for non-stationary, identically distributed, multivariate time series – the sort common in scale-out systems. The incremental update rule and greatly reduced data size allow processing on a single node. The framework holistically compares entire time series, rather than single values. Though originally designed for scale-out systems, it is useful for any outlier detection task where multi-dimensional time series are compared across a time window. The only restriction on the data is that at any single point in time, non-outlier time series are expected to behave similarly.

2. An online, communication-efficient distributed variance monitoring algorithm. It continuously provides an estimate of the global variance of a sliding window to all participating nodes, but avoids unnecessary transmissions. The estimation is within guaranteed (non-probabilistic) user-specified approximation bounds, because it is based on the entire global data set rather than on a sample. As far as we are aware, this is the first such distributed variance estimation scheme.

Each algorithm has a single parameter that directly controls the communication-accuracy trade-off.

We have evaluated the adapted latent fault detector using log data of 110 nodes from a live, real-world system in production. Our simulations show that the adapted
outlier detector is able to reduce bandwidth to 13% of the original’s with no additional false positives, or to 11% with a false positive rate below 1%.

2.1 Latent Fault Detection

In previous work [GSBB12] we presented a statistical latent fault detection framework. Full discussion of the problem, our assumptions and the solution can be found there. This section is a summary of that work, focusing on the sign test.

2.1.1 Problem Description

There are $M$ machines, preforming identical tasks, each periodically\(^1\) reporting $C$ aggregated performance counters in a time window of length $T$. We denote by $x(m, t)$ the vector of counter values for machine $m$ at time $t$, and by $x(t) = \bigcup_m x(m, t)$ their union. We aim to find machines that behave differently.

We begin with a reasonable assumption: in a large system, most machines perform well most of the time. Furthermore, in a scale-out system with load balancing, we expect similar machines with similar hardware and software\(^2\) to exhibit roughly similar behavior as measured by aggregated performance counters (assuming the aggregation interval is sufficiently longer than typical transaction times). The null hypothesis is that the inspected machine is working properly and hence the statistical process that generated $x(m, t)$ is the same statistical process that generated the vector for any other machine $m'$. Thus each counter is identically distributed across machines. Formally, we assume that $x(m, t)$ is a realization of a random variable $X_t$ whenever machine $m$ is working properly. However, if the time series for machine $m$ is notably different from those of other machines, we reject the hypothesis and flag $m$ as suspicious, meaning we suspect it manifests a latent fault. Note that we do expect to see changes over time, due to changes in the workload, for example. Thus we do not commit ourselves to the assumption that the process is stationary; $X_t$ and $X_{t'}$ need not be related in any way, maintaining generality. However, we expect these changes to be similarly reflected in all machines.

We can now formally describe the outlier detection problem. Given the last $T$ data points of $M$ multivariate time series $x(m, t)$ of dimension $C$, and assuming that $\forall t : x(m, t) \sim X_t$, flag each time series (as a whole) as either normal or outlier, with confidence level $0 < \alpha < 1$.

2.1.2 Centralized Latent Fault Detection Framework

Let $S(m, x(t))$ be a test, a ranking function that assigns an outlier score (either a scalar or a vector) to machine $m$ at time $t$. Given a test $S$, and a significance level $0 < \alpha < 1$,

---

\(^1\) We use a sampling period of 5 minutes, a good compromise between responsiveness and transmission delay.

\(^2\) Common in many systems and datacenters [KTGN10, OAS08, KGN08].
we can present the framework as follows:

1. Preprocess: select counters and scale to unit variance.
2. Compute for every machine $m$ the vector:
   $$v_m = \frac{1}{T} \sum_t S(m, x(t))$$ (integration phase).
3. Compute the p-values (defined below) $p(m)$ from $v_m$.
4. Report every machine with $p(m) < \alpha$ as suspicious.

Essentially, the scores for machine $m$ are aggregated over time, so that eventually the norm of the aggregated scores converges and is used to compute a p-value for $m$. The p-value for a machine $m$ is a bound on the probability that a random healthy machine would exhibit such aberrant counter values. If the p-value falls below a predefined significance level $\alpha$, the null hypothesis is rejected, and the machine is flagged as suspicious.

In [GSBB12] we derived and evaluated 3 different tests within the framework. What follows is a summary of the sign test.

### 2.1.3 The Sign Test

The sign test extends the classic statistical sign test to allow the simultaneous multivariate comparison of multiple machines. The “sign” of a machine $m$ at time $t$ is the average direction of its vector $x(m, t)$ to all other machines’ vectors, and its score vector $v_m$ is the sum of all these directions, divided by $T$. The intuition is that healthy machines are similar on average, and any differences are random. Average directions are therefore random and tend to cancel each other out when added together, meaning $v_m$ will be a relatively short vector for healthy machines. Conversely, if $m$ has a latent fault, then some of its metrics are consistently different from healthy machines, and so the average directions are similar in some dimensions. When summing up these average directions, these similarities reinforce each other and therefore $v_m$ tends to be a longer vector.

Formally, the sign test scoring function is

$$S(m, x(t)) = \frac{1}{M-1} \sum_{m' \neq m} \frac{x(m, t) - x(m', t)}{\|x(m, t) - x(m', t)\|}$$ \hspace{1cm} (2.1)

If most machines are working properly, we expect this value to be small, since directions tend to be random. Moreover, the sum of several samples over time is also expected to stay close to zero. Therefore the norm of $v_m = \frac{1}{T} \sum_t S(m, x(t))$ should not be much larger than its empirical mean. The p-value $p(m)$ controls this statistic by guaranteeing a small number of false detections, depending on the significance level $\alpha$. Algorithm 2.1 shows the sign test integrated into the framework. Derivation of $p(m)$ is described in [GSBB12].
Algorithm 2.1: The sign test.

1. foreach machine $m$ do
2. \[ S(m, x(t)) \leftarrow \frac{1}{M-1} \sum_{m' \neq m} \frac{x(m,t) - x(m',t)}{\|x(m,t) - x(m',t)\|} \]
3. \[ v_m \leftarrow \frac{1}{T} \sum_t S(m, x(t)) \]
4. \[ \hat{v} \leftarrow \frac{1}{M} \sum_m \|v_m\| \]
5. foreach machine $m$ do
6. \[ \gamma \leftarrow \max(0, \|v_m\| - \hat{v}) \]
7. \[ p(m) \leftarrow (M + 1) \exp\left(-\frac{TM^2}{2(M+2)^2}\right) \]
8. if $p(m) \leq \alpha$ then report machine $m$ as suspicious

2.2 Distributed Online Variance Monitoring

The framework in Section 2.1.2 requires that counter values across the time window be normalized during preprocessing: each counter should be transformed to zero mean and unit variance\(^3\). In some settings mean and variance are relatively constant or predictable. However, we prefer to avoid that assumption, and handle unpredictable counters.

We use the safe zones approach [KSSL12, SSK08, KSA+14, SSK07b] to monitor both the global mean and the global variance of each counter. In this approach, each monitored machine (node) receives a local constraint on its data $x(m,t)$ from a coordinator node, such that if all local constraints are satisfied, the value for some function $f$ of the global average of $x(m,t)$ is within a pre-defined threshold. In our case, it means that global mean and variance are known to be “close enough” to their last known values. These last known values are then used to normalize the counter values at each node. We can trade-off accuracy and communication by adjusting the thresholds. Violations are less likely if global mean and variance are allowed to drift further from their last known values – reducing communication but compromising accuracy.

The variance of each counter $X$ is monitored independently. Each node maintains a local statistics vector $V_i$ of its last $T$ samples of $X$, and a record of the last known global statistics vector $V(0)$. The current estimate of global variance (and mean) is calculated from $V(0)$. Nodes define lower and upper variance thresholds, e.g. 0.5 and 2 times the current estimate, and derive constraints on $V_i$. Violations are reported to the coordinator, which then polls each node for its current $V_i$ and distributes a new global $V(0)$ to all nodes. Nodes recalculate constraints and monitoring resumes. The scheme is described in detail below and in Algorithm 2.2.

Unlike sampling approaches, our scheme guarantees that variance is within the approximation bounds. Moreover, our solution is able to avoid communication entirely in most rounds. Indeed, if the variance is relatively fixed, our scheme virtually stops communicating after a few initial rounds.

\(^3\) We select counters in advance, using the method described in [GSBB12].
Algorithm 2.2: Online Variance Monitoring.

1 **Initialization**: Run synchronization once.

3 **Node i at time point t:**
4 Predict reference point $V'(t)$ and update safe zone $G$.
5 if $V(0) + d_i(t) \notin G$ then
6     Violation: send local vector $V_i(t)$ to coordinator.
7     Wait for new global reference point $V(0)$.
8     Update prediction variables $V(-\Delta t), \Delta t, V'(t)$.
9     Estimate global variance from $V'(t) = (\mu', \sigma')$: $\sigma' - \mu'^2$.

11 **Coordinator at time point t:**
12 if violation for counter $c$ then
13     Poll all nodes for their $V_i(t) = (\mu_i, \sigma_i)$.
14     Update slacks $\beta_i$ and send to participating nodes.
15     Distribute new global reference point $V = (\mu, \sigma)$.

2.2.1 Notation

The set of values of counter $X$ over the last $T$ times and across all $M$ nodes is denoted by $X(t)$, and $X_i(t)$ denotes the last $T$ values of $X$ at node $i$. The local mean $\mu_i = E[X_i(t)]$ is the mean of the last $T$ values of $X$ at node $i$, while the global mean $\mu = E[X(t)]$ is the mean of the last $T$ values across all nodes. Similarly, denote $\lambda_i = E[X_i(t)^2]$, the local mean of the squares, and $\lambda = E[X(t)^2]$ the global mean of the squares. Let $V(t) = (\mu, \lambda)$ and $V_i(t) = (\mu_i, \lambda_i)$ be the global and local statistics vectors, respectively.

2.2.2 Defining Safe Zones

We wish to estimate the global variance $\text{Var}(X)$ at each time $t$. Recall that:

$$\text{Var}(X) = E[X^2] - (E[X])^2 = \lambda - \mu^2.$$  

We therefore monitor the constraints $L \leq \lambda - \mu^2 \leq H$, for some lower and upper variance thresholds $L$ and $H$. The *admissible region*, the region where constraints hold, is therefore the area between two parabolas. Following [KSSL12], we aim to find a convex safe zone $G$ which is contained within the admissible region. Convexity plays a crucial role in the monitoring process. Since convex sets are closed under averaging, when all local vectors are inside the safe zone, the global mean is guaranteed to be inside as well.

Let $t = 0$ be the last global synchronization time, and let $V(0) = (\mu(0), \lambda(0))$ be the reference point, the last known global mean and mean-of-squares, computed that time. For each node $i$ we define the local drift vector $d_i(t)$ as the change in the current vector from the node’s vector during the last synchronization: $d_i(t) = V_i(t) - V_i(0)$.

Since we wish to monitor that the global $V(t)$ is within some convex set $G$, we
Figure 2.1: Global, local, reference, and drift vectors with two nodes. Note that $V(t) = \frac{V_1(t) + V_2(t)}{2} = \frac{W_1(t) + W_2(t)}{2}$.

define equivalent local constraints on the drift vectors. The current local vectors can be written in terms of drift vector $d_i$: $V_i(t) = V_i(0) + d_i(t)$. Note that the global vector is the mean of the local vectors, and can thus be written as the mean of drifts and the reference point:

$$V(t) = \frac{1}{M} \sum_i V_i(t) = V(0) + \frac{1}{M} \sum_i d_i(t). \quad (2.2)$$

Let $W_i(t) = V(0) + d_i(t)$ be the local drift from the last reference point. Note that $V(t) = \frac{1}{M} \sum_i W_i$, recall $G$ is convex, and from (2.2) we arrive at the local constraints: if $\forall i, W_i \in G$ then $V(t) \in G$. Figure 2.1 illustrates these concepts for two nodes.

We derive two separate safe zones: one for variance above $L$ and another for variance below $H$. As long as $W_i$ is inside both safe zones in all nodes, we are guaranteed that the variance is within the allowed range.

**Variance Above Lower Threshold** We wish to define a convex safe zone $G_L$ such that as long as $V(t) \in G_L$ then $\text{Var}(X) \geq L$. This corresponds to monitoring that $\lambda - \mu^2 \geq L$, which is already a convex set – the area above a parabola – and can be directly used as safe zone. Therefore the local constraint for each node $i$ is simply $W_i(t) \in G_L$. Let $W_i(t) = (a, b)$ and monitor that $b - a^2 \geq L$.

**Variance Below Upper Threshold** We wish to define a convex safe zone $G_H$ so that as long as $V(t) \in G_H$ then $\text{Var}(X) \leq H$. The area below a parabola is not a convex set. However, we can find a tangent half-plane $I$ below this parabola. $I$ is a convex set, and since $I \subset G_H$, then as long as $V(t) \in I, V(t) \in G_H$ and therefore $\text{Var}(X) \leq H$.

We use the reference point $V(0)$ to find the optimal hyperplane. The thresholds $H$ and $L$ are reset during synchronization, so obviously $V(0) \in G_H$. We can choose any half-space $I$ such that $V(0) \in I$, but to avoid unnecessary future synchronization we choose $I$ such that $V(0)$ is far from the boundary of $G_H$. Doing so ensures that

---

We revisit this technique later in Section 3.2.1, and generalize it in Lemma 3.2.1.
drift has to be large to cause a violation. Consequently, we choose $I$ as the tangent at point $P$, where $P$ is the closest point to $V(0)$ on the parabola $\lambda - \mu^2 = H$, and the local constraint is $W_i \in I$. We can find $P$ by solving a cubic equation minimizing the distance from the parabola to $V(0)$. For example, if $V(0) = (0.5, 1)$ and $H = 1.5$, then the closest point on the parabola is $\mu \approx 0.237$. This yields the point $P = (0.237, 1.556)$, and the induced safe zone $I$: the half-plane $\lambda - 0.474\mu < 1.443$. Figure 2.2(a) shows $V(0)$, $P$ and the resulting safe zone, and Figure 2.2(b) shows the intersection with the safe zone for the lower limit $L = 0.5$.

2.2.3 Bounding Variance Approximation Error

During each synchronization we recompute the new safe zone around the reference point. Large safe zones mean less communication, but they also allow the true variance to deviate further from the global estimate, increasing error. Proper threshold selection helps us bound the approximation error. We provide a multiplicative approximation bound that controls relative error, as it is more suitable for normalization. Additive bounds for absolute error are also possible.

Let $\sigma_0^2$ be the last known global variance, given the current reference point $V(0)$:

$$\sigma_0^2 = \text{Var}(X(0)) = \lambda(0) - \mu(0)^2,$$ and denote the current global variance $\sigma^2 = \text{Var}(X) = \lambda - \mu^2$. We normalize counter data using the last known standard deviation $\sigma_0$ rather than $\sigma$. We set the the lower threshold to $L = \frac{\sigma_0^2}{f^2}$ and the upper threshold to $H = f^2\sigma_0^2$, where $f > 1$ is a constant that determines the allowed deviation. These thresholds ensure that $\frac{\sigma_0^2}{f^2} < \sigma^2 < f^2\sigma_0^2$ as long as there is no safe zone violation, and therefore the standard deviation used to scale the counter is bounded by $\frac{\sigma_0}{f} < \sigma < f\sigma_0$.

2.2.4 Safe Zone Violations

If one of the local constraints $W_j \in G$ is violated, it may be because $\text{Var}(X)$ is no longer in the range, or due to a false alarm. The simplest way to deal with a violation is to
perform a global synchronization: each node sends its current $V_i(t)$ (the two values $\mu$ and $\lambda$) to the coordinator. The coordinator resets the time to $t = 0$, computes the new global reference point $V(0)$, and sends it to the nodes. These synchronizations also improve estimation accuracy, since the nodes have fresh global mean and variance.

However not all violations are the same. We describe three types of possible violations: true violations, global violations and local violations. A true violation is when the global aggregate $V(t)$ is outside the admissible region (variance larger than $H$ or smaller than $L$), and therefore synchronization must be performed. A global violation happens when the global aggregate is outside the safe zone, but still inside the admissible region (recall that the safe zone is a convex subset of the admissible region). In this case, synchronization is unavoidable but is not strictly necessary, suggesting that the shape of the convex safe zone is sub-optimal. A local violation is when the global aggregate is inside the safe zone, but the local vectors of one or more nodes are outside their local safe zone. Ideally, we would like to avoid these violations since they trigger unnecessary additional synchronizations that increase communication. We explore two approaches to reduce the number of spurious safe zone violations.

### 2.2.5 Multiplicative Slack Distribution

The safe zones described in Section 2.2.2 are uniform – all nodes share the same safe zones. One well-known way to reduce violations would be to allocate different “slack” to each node to exploit its local statistics [Cor13, BO03], in our case by giving each node a different safe zone. Naturally, we must still preserve the guarantee that if all local node vectors are in their respective safe zones, then the global aggregate is inside the admissible region.

The local safe zones $G_i$ assigned to each node $i$ are simply scaled versions of the original uniform safe zone $G$, centered as before around $V(0)$. Each node has a local slack factor $\beta_i$. When testing for local violation, node $i$ first scales its drift vector by $\frac{1}{\beta_i}$: $W_i(t) = V(0) + \frac{1}{\beta_i}d_i(t)$, effectively scaling the safe zone by $\beta_i$. The global drift is now a weighted mean of local drift vectors, rather than a simple arithmetic mean. We preserve our global guarantee by making sure local slacks factors always sum to $M$. Here we again exploit the convexity of $G$. Since $G$ is convex and $\sum \beta_i = M$, then $F = \left\{ \frac{1}{M}\sum z_i | z_i \in G_i \right\} \subseteq G$ (known as the Minkowski mean). Thus the convex hull of local drifts is still inside $F$, and therefore the global aggregate is inside the safe zone $G$.

Each node begins with a slack of $\beta_i = 1.0$. At each round, let $K$ be the set of nodes that reported a local violation to the coordinator this round. If $|K| \geq M\frac{4}{M}$, we assume that current slacks are inadequate, and reset all slacks to 1.0. Otherwise, let $\mathcal{L}$ be a set of balancing nodes (defined below). We take slack from the set of balancing nodes and distribute it to violating nodes. Inspired by LRU cache policies, we choose the nodes with the least number of slack operations (fewest local violations and slack balancing contributions), since they are the most likely to have available...
slack. Thus we define $L$ as the set of $3|\mathcal{K}|$ nodes with the least number of slack operations, excluding nodes in $\mathcal{K}$. Balancing nodes have their slack decreased by a constant factor $w > 1.0$: $\forall i \in L : \beta'_i \leftarrow \frac{\beta_i}{w}$. This extra slack is distributed to violating nodes: $\forall i \in \mathcal{K} : \beta'_i \leftarrow \beta_i + \frac{\gamma}{|\mathcal{K}|}$, where $\gamma = (1 - \frac{1}{w}) \sum_{i \in L} \beta_i$ is the slack gained from balancing nodes. Finally, the coordinator sends the new slacks $\beta'_i$ to the nodes in $\mathcal{K}$ and $L$.

Our slack allocation scheme is motivated by several observations. First, we observe that in our setting, counters of healthy nodes are assumed to be identically distributed, and therefore their direction to the reference point is random. Simply translating the safe zone does not reduce local violations. This phenomenon was also observed in practice during preliminary experiments. Second, we do assume a small number outlier nodes, whose counters might cause frequent local violations. Over time, our balancing scheme distributes more slack (meaning larger safe zones) to such outlier nodes, balanced across the greater number of healthy nodes. Finally, this scheme entails sending just one extra value per participating node, meaning it has the smallest impact on communication.

2.2.6 Reference Point Prediction

Section 2.2 describes a static scheme: nodes always use the last known global reference point $V(0)$ to monitor and estimate variance. If nodes are able to correctly predict how the reference point changes over time, they can adjust the safe zone accordingly, thus reducing safe zone violations and communication costs. The prediction model must ensure that all nodes make the same prediction, as it is used for monitoring and estimation. Prediction models have been shown to be effective in reducing communication [GDG+12, CGMR05]. Prediction is most effective when the data does not change too quickly. However, even if data behavior constantly changes, the safe zone technique guarantees correctness; prediction will simply be less effective.

We incorporate a simple linear prediction model assuming constant “velocity”. Given current reference point $V(0)$, let $\Delta t$ be the time difference between the last global synchronization ($t = 0$) and the previous synchronization, and let $V(-\Delta t)$ be the reference point at that time. The predicted reference point at current time $t$ is simply:

$$V'(t) = V(0) + t \frac{V(0) - V(-\Delta t)}{\Delta t}.$$ 

The prediction $V'(t)$ is then used to compute the current variance estimate and derived safe zone as described above.

Observe that each node can store $\Delta t$, $V(0)$ and $V(-\Delta t)$ during normal operation, and so this model requires no extra communication. Moreover, the model only depends on global data made available during synchronization, and so it yields the same prediction on all nodes.

Since we allow the predicted reference point and the resulting safe zone to move outside the original safe zone, we are no longer guaranteed that $\frac{\sigma_0}{f} < \sigma < f\sigma_0$. Instead,
Algorithm 2.3: Adapted Latent Fault Detector.

1 Initialization:
2 Start variance monitoring on each counter.
3 
4 Node i at time point t:
5 foreach counter c do
6     Sample $X_c(t)$, the value of counter c at time t.
7     Update variance monitor for counter c with $X_c(t)$.
8     Scale $X_c(t)$ using estimated global mean and variance.
9     Let $x$ be the vector of normalized counter values.
10 Compute sketch of $x$ and send to coordinator.
11 
12 Coordinator at time point t:
13 Handle violations in all variance monitors.
14 Receive sketches from all nodes.
15 Compute test function $S$ on received sketches.
16 foreach machine m do
17     Add most recent test function results to $v_m$.
18     Subtract least recent test function results from $v_m$.
19     Update $v_m$ sliding window.
20     Calculate p-value $p(m)$ and flag if $p(m) \leq \alpha$

our safe zones guarantee that $\sigma' / T < \sigma < f \sigma'$, where $\sigma'$ is the standard deviation from the predicted reference point $V'(t)$.

2.3 Distributed Outlier Detection

Given the normalized counter, we now describe an online, communication-efficient version of the latent fault detector summarized in Section 2.1. The original algorithm requires that nodes send all measurements to a central location: $T$ samples of $C$ counters for each of the $M$ machines. Beyond bandwidth and storage costs, processing so much data in a timely manner is difficult on a single machine, due to its size and high dimensionality.

We address these issues using sketching [Mut05, CG07], a common technique for processing large data streams without having to send, store and process all the data. Sketching reduces the size of the data, while still enabling queries. For our purposes, a sketch is a summary function that takes a vector and transforms it to a much smaller vector while approximately preserving some desired property, such as Euclidean distances [JL84] or frequency moments [AMS99]. Beyond reducing the communication load, sketching has the added benefit of reducing the computational load, since the dimensionality of the data is greatly reduced. Finally, we provide an incremental update rule yielding an online adaptation of the framework with reduced memory and processing requirements. The pseudocode is shown in Algorithm 2.3 and explained in detail below.
2.3.1 Sketches

We use sketches to greatly reduce the amount of data sent from each machine and processed by the coordinator. Instead of sending all counters, each node calculates a sketch of the said counters and sends only that. For example, 200 counters could be reduced to 10 dimensions, achieving an immediate 95% reduction in size. The coordinator (or monitoring node) then performs outlier detection using the sketches, rather than the original data.

Formally, rather than apply the test ranking function \( S \) to the set of all local counter vectors \( x(m,t) \), each machine \( m \) will first apply a sketching function \( g \) to its vectors, and send only the sketch \( \hat{x} = g(x(m,t)) \) for processing. The adapted test \( \hat{S} \) will be applied to the sketches rather than the original vector: \( v_m = \frac{1}{T} \sum_t \hat{S}(m, \hat{x}(t)) \). Depending on \( S \), we might also need to adapt the matching p-value calculation.

One well-suited sketching technique is random subspace embedding, which involves a random linear projection to \( k < C \) dimensions. In our setting, each machine projects its counter vectors to \( k \) dimensions using a suitably constructed projection matrix: \( \hat{x}(m,t) = g(x(m,t)) = Rx(m,t) \) where \( R \) is a random \( k \times C \) matrix constructed as described in [JL84, Ach03]. The Johnson-Lindenstrauss Lemma [JL84, Ach03, Mat08] shows that such random embeddings preserve inter-point distances with bounded distortion in high probability. This also makes the sketch general enough that the same sketch can be used as input to different tests. We show below that the sign test can be applied directly to the sketched vectors.

2.3.2 Sign Test on Linear Sketches

We first offer an intuitive, geometric explanation. The sign test function (2.1) from Section 2.1.3 depends only on the normalized direction from \( x(m,t) \) to the other vectors. Put differently, the result depends only on the distribution of directions on the unit sphere centered on \( x(m,t) \) – the angles between directions. Given the assumptions in Section 2.1.2, for healthy (normal) machines the normalized directions to other machines tend to be distributed spherically symmetric the sphere. In other words, there is 180-degree symmetry around the center – in each dimension (counter) high values from some machines are balanced by low values from others. The adapted sign test is the sum of normalized directions from \( x(m,t) \) after the transformation \( R \), which preserves these angles with little distortion. In a sense, \( R \) “rotates” unit vectors around \( x(m,t) \) without changing angles. Finally, the sign test p-value does not depend on the dimensionality of the vectors. Therefore we can apply the sign test directly to the sketched vectors \( \hat{x}(m,t) \).

Bounded Distortion

We begin by showing we can apply the sign test function \( S \) to the sketched vectors because \( R \) does not overly distort the original vectors. Note \( S \) depends only on the normalized
direction from $x(m, t)$ to the other vectors. Since the vectors are normalized, the length of the sum $\| \frac{1}{T} \sum_t S(m, x(t)) \|$ is entirely determined by the angles between the vectors, i.e. their inner products: $\|u + v\|^2 = \|u\|^2 + \|v\|^2 + 2\langle u, v \rangle = 2 + 2\langle u, v \rangle$. Similarly, the adapted sign test normalizes the directions from $x(m, t)$ after the transformation $R$: $\| \frac{1}{T} \sum_t S(m, Rx(t)) \|$, so it too is determined by the inner products of unit vectors.

Consider a simple random unitary matrix – a rotation. Clearly rotating unit vectors before addition does not affect the length of the result. More generally, if the sketching matrix $R$ preserves inner product of unit vectors, the length of the sum of normalized sketched vectors is equal to the original length. Therefore it is sufficient to show that $R$ preserves inner products of unit vectors with high probability.

**Lemma 2.3.1.** Given $\epsilon \in (0, \frac{1}{2})$ and $\delta \in (0, 1)$, let $k = O(\log \frac{k}{\epsilon^2})$ a large integer and $R \in \mathbb{R}^{k \times C}$ a suitable random projection (as in [JL84, Ach03]). Then with probability $1 - \delta$, for any two unit vectors $u, v \in \mathbb{R}^C$, 

$$|\langle Ru, Rv \rangle - \langle u, v \rangle| \leq 2\epsilon.$$ 

**Proof.** From a variant of the Johnson-Lindenstrauss lemma [Mat08, Theorem 3.1], with probability $1 - \delta$, for any single arbitrary vector $x \in \mathbb{R}^C$, 

$$(1 - \epsilon)\|x\|^2 \leq \|Rx\|^2 \leq (1 + \epsilon)\|x\|^2 .$$

(2.3)

Recall that $\|u - v\|^2 = \|u\|^2 + \|v\|^2 - 2\langle u, v \rangle = 2 - 2\langle u, v \rangle$, and similarly $\|Ru - Rv\|^2 = \|Ru\|^2 + \|Rv\|^2 - 2\langle Ru, Rv \rangle$. From (2.3) we know $\|Ru\|^2 \leq (1 + \epsilon)\|u\|^2 = (1 + \epsilon)\|u\|^2$ and $\|Rv\|^2 \leq (1 + \epsilon)\|v\|^2$. Thus, 

$$\|Ru - Rv\|^2 \leq 2(1 + \epsilon) - 2\langle Ru, Rv \rangle .$$

(2.4)

Applying (2.3) to $x = u - v$: 

$$\|Ru - Rv\|^2 \geq (1 - \epsilon)\|u - v\|^2 = (1 - \epsilon)(2 - 2\langle u, v \rangle) .$$

(2.5)

Combining (2.4) and (2.5) and rearranging, we get 

$$\langle Ru, Rv \rangle - (1 - \epsilon)\langle u, v \rangle \leq 2\epsilon$$

and because $\epsilon < \frac{1}{2}$, 

$$\langle Ru, Rv \rangle - \langle u, v \rangle \leq \langle Ru, Rv \rangle - (1 - \epsilon)\langle u, v \rangle \leq 2\epsilon .$$

The other direction is analogous. 

Given $M$ machines, there are $O(M^2)$ pairs of $u, v$. Choosing $k = O(\log(M)/\epsilon^2)$ and applying the union bound for all such pairs will guarantee low distortion ($\leq 2\epsilon$) with
high probability \((1 - \delta, \text{ where } \delta = O(1/M^2))\).

Note that the distortion introduced by the sketch \((\epsilon)\) grows very slowly with number of machines: for fixed communication \(k\), the distortion is \(\epsilon = O\left(\sqrt{\log M/k}\right)\).

### Computing p-values

Using machinery from [GSBB12], we now prove that the p-values for the adapted test \(\hat{S}\) can be computed the same way as the regular test.

**Proof.** The sign test ranking function \(S\) from Section 2.1.3 is \(2, \frac{2}{M-1}\)-bounded\(^5\): if we change all counter values, a machine score cannot change by more than 2; if we change the counter values for a single machine, the score for any other machine cannot change by more than \(\frac{2}{M-1}\).

Since \(\hat{S} = S(m, \hat{x}(t))\), and \(\hat{x}(m, t) = Rx(m, t)\) independently for each \(m\), then it follows that \(\hat{S}\) is also \(2, \frac{2}{M-1}\)-bounded. We can therefore apply Lemma 2 from [GSBB12], yielding the same p-value in Algorithm 2.1. \(\square\)

#### 2.3.3 Online Integration Using a Sliding Window

The integration phase in stage 2 of the framework in Section 2.1.2 computes \(v_m = \frac{1}{T} \sum_t S(m, x(t))\). Computing \(S(m, x(t))\) only requires the data from time \(t\). Thus we can adapt any test into an online test by maintaining a sliding window per machine of test function \((S)\) outputs for the last \(T\) sketches. When new data arrives at time \(t\), the coordinator updates the current \(v_m\) of each machine by computing and adding \(\frac{1}{T} S(m, \hat{x}(t))\), and subtracting the least recent stored test result, \(\frac{1}{T} S(m, \hat{x}(t - T - 1))\).

The p-value can then be computed from \(v_m\) in the usual manner. Since \(S\) is now computed only for the most recent time, and since the sketches are of low dimension \(k\), processing and memory costs are low. For the sign test, the runtime for incremental update is \(O(kM^2)\) and the sliding window requires \(O(kMT)\) memory. The reduced size allows the computation to be done on a single coordinator machine on time, before the next round starts. For example, for the large system evaluated in [GSBB12] \(M = 4500\), \(T = 288\), and with \(k = 10\), the update rule is easy to compute on a single machine within the sampling period of 5 minutes.

#### 2.4 Evaluation

Since the efficacy of the latent fault detector has already been established in previous work [GSBB12], we evaluate the communication-efficient online adaptation by comparing it directly to the centralized offline detector. Using a random subset of the real world dataset from [GSBB12] (counter logs from the index service of a large search engine), we performed a series of simulations to explore the behavior of the adapted algorithm.

---

\(^5\)Definition from [GSBB12]
During simulation we run through the counter logs, simulate the operation of nodes and the coordinator, and keep track of communication. The dataset consists of counter logs for $M = 110$ randomly selected machines, each reporting $C = 216$ counters that were automatically selected during the pre-processing phase as described in [GSBB12]. The window length was set to $T = 144$. Unless otherwise noted, our simulation includes the slack and reference point prediction mechanisms described in Section 2.2. The constant slack factor $w$ was set to $1.75$ (1.1 when not using prediction) by tuning on a subset of the data.

We evaluate performance with three metrics. We do not include the first $T - 1$ initialization rounds in these metrics; evaluation starts from the first full window of $T$ samples.

1. **Communication fraction**: number of floating point values sent by the system, divided by $C \cdot M \cdot T$, the centralized cost of sending $C$ counters from each machine at each round. Note that we count the total number of discrete values, rather than number of messages sent. We include all sent values: those sent by the nodes, the coordinator, and the sketched values sent for the test.

2. **Error**: mean absolute difference between the order of magnitude of machine p-values in the centralized sign test and its communication-efficient adaptation. Formally, let $p(m)$ and $p'(m)$ be the mean p-values assigned to machine $m$ across all time windows in the centralized and adapted algorithm, respectively; the error measure is: $\frac{1}{M} \sum_{m \in M} \log_{10} p(m) - \log_{10} p'(m)$. We take the log of the p-values because significance thresholds are usually set based on magnitude, for example $0.01$ or $0.0001$.

3. **Detection error**: percentage of classification differences across all machines and time windows. This measures the ability of the adapted test to give identical classifications (outlier or normal) as the centralized test. We set the significance level to $\alpha = 0.01$.

### 2.4.1 Performance

To illustrate the trade-off between communication and accuracy, we performed a parameter sweep over several values of sketch size $k$ and safe zone size (threshold factor) $f$. The results are shown in Figure 2.3.

Figure 2.3(a) shows the fraction of communication performed for each $k, f$ combination. Sketch size $k$ has the largest effect on communication. Minimal communication (11%) is achieved when $k = 5, f = 10$ with an error (as defined above) of 0.122, while minimal error (0.033) is achieved at non-realistic $k = 216$ (no sketching), $f = \sqrt{2}$ with communication cost of 160%. Turning off sketching ($k = C = 216$) serves as a lower (if impractical) bound on the error. Communication fraction is greater than 1.0, since in addition to variance monitoring overhead, each node sends 216 counters at each round. With sketching applied, communication drops to 10–70%, depending
on the exact values of \( k \) and \( f \). Very tight safe zones \( (f \in \{\sqrt{2}, 2\}) \) also result in an increase in communication. When the safe zone scaling factor is more permissive \( (f \geq 3) \) monitoring cost is relatively constant, and communication due to larger sketch sizes dominates communication cost. Figure 2.3(b) shows the resulting error (mean difference between magnitude of average machine p-value). Smaller sketches induce greater errors, while larger safe zones increase error but have smaller effect.

One interesting observation is that more permissive safe zones \( (f \geq 3) \) do not result in substantially lower communication. We attribute this phenomenon to the effectiveness of the reference point prediction. We repeated the experiments with the reference point prediction mechanism disabled, and show the results in Figure 2.4. Communication costs were more predictable: both sketch size \( k \) and safe zone size \( f \) affect communication. Increased safe zone size always means less communication, with the minimum achieved as expected for the largest safe zone factor \( f = 12 \) and the smallest sketch size \( k = 5 \). Error behavior was also clearer: it is most affected by sketch size, and the anomalous error peak around \( f = 4, k = 20 \) disappeared. We further investigate the effects of prediction in Section 2.4.2.

Figure 2.3(c) depicts the practical effect of different \( k, f \) combinations on the latent fault detector. Recall that detection error is defined as the percentage of machine classifications (normal, outlier) that differ between the centralized approach and the adapted, communication-efficient online approach. We’ve seen that \( k \) dominates p-value errors; this is confirmed by Figure 2.3(c), which shows detection error for each \( k, f \)
configuration. For \( k \geq 10 \) there are few to no detection errors. We conclude that the adapted latent fault detector is robust. It matches the centralized approach very well despite an order of magnitude reduction in communication. Moreover, the adapted detector is resilient to non-uniform scaling, performing well even with permissive \( f \).

### 2.4.2 Slack and Reference Point Prediction

Section 2.2 describes slack and reference point prediction, two techniques to reduce communication, though with a possible increase in error. Figure 2.5 explores how they affect performance, using different variations of the algorithm.

**Basic** Without slack and reference point prediction.

**Slack** Use slack with \( w = 1.1 \), without prediction.

**Prediction** Use reference point prediction but no slack.

**Both** Use both slack \((w = 1.75)\) and prediction.

Figure 2.5(a) compares the four variants across different \( f \) values, with their median marked in a red line to represent typical behavior across the range of values. We focus on the impact of variance monitoring, so sketching is turned off \((k = C = 216)\) and the relevant communication is not counted. Slack has only modest contribution, because savings from avoided local violations are offset by slack factor distribution. Indeed the best improvement is produced by reference point prediction, which requires no additional communication. Reference point prediction reduces the overhead from 30\% to just over 10\%. This reduction, however, comes at a price: reference point prediction increases the error, as Figure 2.5(b) shows. This is both because there are fewer synchronizations, and because we allow the predicted reference point to move outside the original safe zone. Bad reference point predictions can introduce additional error (though still within approximation bounds). Conversely, slack has very small impact on accuracy.

Note that when very tight safe zones \((f < 3)\) are used with prediction, there is a slight increase in communication. This is because there is simply not enough slack available.
to offset the extra cost of slack distribution when violations occur. This illustrates the
difference between measuring the number of transmitted messages, common in some
settings, and the total size of transmitted messages.

Recall the three types of safe zone violations possible in the monitoring algorithm:
true violation, global violation and local violations. To assess the effectiveness of the
two violation prevention mechanisms, we plot the average number and composition of
different types of violations in Figure 2.5(c) with $f = 10$. Global violations are not
plotted, because they are so rare as to be invisible. Slack distribution and reference
point prediction are designed to reduce synchronization by reducing the number of local
violations. By itself, slack reduces the average number of violations to 83% of the basic
variant, while reference point prediction reduces it to 39%. Combined, violations are
reduced to 27%. Furthermore, with the basic variant over 99% of all violations are local.
The slack and reference point prediction mechanisms reduce local violations percentage
down to 92.5% of all violations (of which there are fewer), indicating that slack and
prediction mechanisms indeed manage to prevent many local violations.

We find that global violations are almost nonexistent in practice – below 0.04 per
round for all variants. Our experiments show that on average over all $k, f$ combinations,
when not using prediction and slack, only 0.2% of violations are global, increasing up
to 1.5% when using prediction and slack. Furthermore, the average ratio of global
violations to true violations is always below 0.001. This suggests that our choice of
convex safe zone is indeed close to optimal, since practically every time the global
aggregate crosses the safe zone, it also crosses the admissible region.
Chapter 3

Practical Entropy Approximation for Distributed Streams

The Shannon entropy of a discrete random variable $X$ taking values $\{x_1, \ldots, x_k\}$ is

$$H(X) = -\sum_{i=1}^{k} \Pr[x_i] \ln \Pr[x_i].$$

Entropy\(^1\), often described as a measure of disorder or information content, is widely used across many scientific and engineering disciplines, with diverse applications such as network monitoring [ZLO+07, NSA+08], time series analysis [BP02, SG02], medicine [And04, KCAS05], neuroscience [NBdRvS04], signal processing [RD01], and anomaly detection [BJS15].

Recent years have seen an explosion in the number of devices and sensors, and with it new applications of entropy: network attack detection [GGSZ14, ABBS10], EEG monitoring [KCAS05, Bru06], air quality monitoring [CKL+04], physical activity detection using wearable sensors [EPMK08], sensor fusion in wireless sensor networks [SL13], and many more. Though new streaming approaches [HNO08, CC13] help tackle the growth in both rate and volume of incoming data, these still assume that data can be centralized – i.e., a single stream.

In the increasingly relevant setting of distributed streams, however, multiple data sources give rise to new challenges. First, entropy is highly non-linear, making accurate entropy estimation from distributed data a hard problem. Second, when input sources are distant or battery-constrained wireless devices, transmitting even small messages can be very costly in terms of battery power [GKD+13, SL13] or increased latency [Cor13]. Hence the need for communication-efficient approaches: algorithms that avoid sending messages.

Existing approaches to distributed entropy estimation can be impractical in these

\(^1\) Entropy is an overloaded term. Unless noted, we use “entropy” as shorthand for the well-known maximum likelihood estimator for Shannon entropy, detailed in Section 3.1.
settings. Sketching \cite{HNO08, CC13} reduces the size of data updates but each such update must still be communicated – impractical when the input rate is high \cite{ABBS10} or sensors are battery powered. *Insertion-only algorithms*\(^2\) \cite{Cor13, ABC09} can reduce the number of messages, but they do not support deletion of previously observed inputs, and so cannot estimate entropy in the recent (sliding) window. *Sampling* approaches \cite{Cor13} limit the number of sent messages per window, but updates are still sent in each window to maintain the sample even when entropy is unchanged. They also require large samples to accurately approximate entropy \cite{Pan03, NBdRvS04, HNO08}. *Periodic sampling* also introduces delays, and the update period must be tuned to balance latency, error, and communication \cite{Cor13}. Finally, the probabilistic bounds of existing approaches could be unsuitable for some tasks, for instance seizure detection \cite{KCAS05}.

In short, existing approaches offer either reduced communication but no support for sliding windows, or continuous communication and probabilistic error bounds.

### This Chapter

We propose CIDER (Communication-efficient Distributed Entropy estimator), a new deterministic distributed entropy approximation algorithm that reduces communication and supports sliding windows, without losing the advantages of previous approaches. Our goal is a practical “turn-key” replacement for many kinds of distributed entropy monitoring applications that currently use centralization. What makes CIDER practical?

- It is communication-efficient: messages are sent only when entropy changes; if the changes are sufficiently small and do not violate the approximation bounds, CIDER can still avoid communication.
- It supports event deletions\(^3\), meaning that it can monitor entropy in a sliding window of fixed or variable size, including full support for time-based windows.
- Approximation bounds are deterministic and user-defined – users set error bounds directly.
- For bandwidth-limited or very high-dimensional problems, it incorporates a state-of-the-art entropy sketch \cite{CC13} to reduce message size and avoid holding explicit counters in memory.
- It provides the current entropy estimate in all nodes, rather than in just a single coordinator node. This can be crucial in settings such as wireless sensor networks \cite{SL13}.

In summary, CIDER is the first communication-efficient distributed entropy approximation with deterministic error bounds that fully supports deletions (specifically the strict turnstile model), fixed-size windows, and time-based windows.

We evaluate CIDER on real-world datasets representing real applications: network monitoring, load balancing, and air quality monitoring. Our results show that CIDER

\(^2\)Also called the cash register streaming model \cite{Mut03}; event counts can only increase.

\(^3\)The strict turnstile streaming model: counts go up and down, but are never negative.
reduces communication by up to two orders of magnitude (compared to centralizing the data) and scales well with the number of nodes.

3.1 Problem Definition and Notation

Let $X$ be a discrete random variable taking values $\{x_1, \ldots, x_k\}$. Then its Shannon entropy is

$$H(X) = - \sum_{i=1}^{k} \Pr[x_i] \ln \Pr[x_i]$$

(we use the natural logarithm, but other bases are also used).

Since $\Pr[X_i]$ is generally unknown, a widely-used estimator is the maximum likelihood estimator $[\text{Pan03}]$, given $b_i$ observations of each $X_i$, and denoting by $n = \sum_{i=1}^{k} b_i$ the total number of observations, the entropy estimate is

$$H_{\text{MLE}}(P) = - \sum_{i=1}^{k} p_i \ln p_i$$

where $p_i = b_i/n$ and $P$ is the vector $[p_1, p_2, \ldots, p_k]$. For a continuous $X$, the widely used histogram estimator is computed in the same way: divide the range of $X$ to $k$ bins and define $\{b_i\}_{i=1}^{k}$ to be the number of observations in each respective bin, $n = \sum_{i=1}^{k} b_i$, and $p_i = b_i/n$ (we also define $p_i \ln p_i = 0$ if $p_i = 0$). In general, the MLE is negatively-biased and requires many samples to control both variance and bias $[\text{Pan03}, \text{NBdRvS04}]$, which is one reason distributed sampling approaches are problematic for entropy estimation $[\text{HNO08}]$: communication grows approximately linearly with sample size $[\text{Cor13}]$. Nevertheless, the “naive” MLE estimator is ubiquitous in practice, and in some cases the bias is unimportant, for example in feature extraction for subsequent supervised learning.

In the continuous distributed monitoring model $[\text{Cor13}]$, there are $m$ nodes, each observing an infinite stream of events $\{(t, i)\}$, where $t$ is the arrival time, and $i \in \{1 \ldots k\}$ is observed event: $X_i$ or a value in bin $i$. Each node $j$ maintains a sliding window holding the last $n^j$ observations seen in the window: in fixed-size windows the number of observations $n^j$ is fixed, while in time-based windows this number changes dynamically and is determined by the number of observations seen during the window interval. For example, $n^j$ for a five-minute window is the number of observations that arrived in the last five minutes, which can be two or two million.

Let the local count $b^j_i$ be the number of observations of type $i$ ($X_i$ or bin $i$) in the local sliding window of node $j$, with local size $n^j = \sum_{j=1}^{m} b^j$. Note that $b^j_i$’s can increase or decrease as observations enter or exit the window, but can never be negative\(^4\). We also define the local probability vector of node $j$, $P^j = [p^j_1, p^j_2, \ldots, p^j_k]$, where $p^j_i = b^j_i/n^j$. The global sliding window is the union of all local sliding windows. It contains $n = \sum_{j=1}^{m} n^j$

\(^4\)Hence this is the strict turnstile streaming model $[\text{Mut03}]$. 33
observations, with \textit{global counts} \( b_i = \sum_{j=1}^{m} b^j_i \). We denote by \( P \) the \textit{global probability vector} \( P = [p_1, \dotsc, p_k] \) where \( p_k = b_i/n \).

We aim to maintain an arbitrarily accurate approximation of \( H_{\text{MLE}}(P) = -\sum_{i=1}^{k} p_i \ln p_i \). At any time, all nodes must provide \( h_0 \): an approximate value of \( H_{\text{MLE}}(P) \), such that \( LB(h_0) < H_{\text{MLE}}(P) < UB(h_0) \), where \( LB(h_0) \) and \( UB(h_0) \) are user-defined error bound functions set by the user, and \( h_0 \) is identical at all nodes.

This formulation can express any additive or multiplicative approximation, with relative or absolute error, by appropriately defining the functions \( LB \) and \( UB \). For example, to maintain a \((1 \pm 0.2)\) approximation, we define \( LB(h_0) = h_0 \times 0.8 \) and \( UB(h_0) = h_0 \times 1.2 \). We do require that \( LB(h_0) < H_{\text{MLE}}(P) < UB(h_0) \).

### 3.2 The CIDER Estimator

Our basic strategy is to convert the entropy approximation problem to a threshold monitoring problem of the form \( H_{\text{MLE}}(P) < T_L \) or \( H_{\text{MLE}}(P) > T_U \). Nodes initially forward their local probabilities to a \textit{coordinator} node, which computes the current \( h_0 = H_{\text{MLE}}(P) \), and thresholds \( T_L = LB(h_0) \), \( T_H = UB(h_0) \); we call this procedure \textit{synchronization}. We then proceed with the following basic algorithm, which guarantees that \( h_0 \) is close to the current value \( H_{\text{MLE}}(P) \):

1. As long as \( T_L \leq H_{\text{MLE}}(P) \leq T_U \): output approximation \( h_0 \) (Alg. 3.1).
2. Otherwise, \textit{sync} and continue with the new \( h_0 \), \( T_L \), and \( T_H \) (Alg. 3.2 or Alg. 3.3).

By using \( h_0 \) as the estimate and continuously monitoring for threshold crossings, we provide an accurate approximation to \( H_{\text{MLE}}(P) \). Since \( h_0 \) is constant, the estimate is identical at all nodes. For example, if \( h_0 \) is currently 3.5 and we aim to maintain a \((1 \pm 0.2)\) approximation, we set \( T_L = h_0 \times 0.8 = 2.8 \) and \( T_U = h_0 \times 1.2 = 4.2 \). Now suppose that we later detect a threshold violation. After synchronizing, we see that \( H_{\text{MLE}}(P) \) is now 4.3. We therefore update \( h_0 = 4.3 \), \( T_L = 3.44 \), \( T_U = 5.16 \), and resume monitoring.

We have reduced the problem to a simpler one: given \( h_0 \) and thresholds \( T_L = LB(h_0) \), \( T_H = UB(h_0) \), we need to detect whenever the global entropy \( H_{\text{MLE}}(P) \) crosses below \( T_L \) or above \( T_U \). \textit{Geometric monitoring} [KSSL12, KSA+14] is a recent framework for communication-efficient monitoring of such threshold queries. In geometric monitoring, we convert the thresholds to constraints on the global probabilities, and decompose these global constraints to local constraints on the individual input streams that can be tested independently at each node. Nodes only communicate when local constraints are violated, in which case the coordinator can resolve the violation, for example by synchronizing as above.

Sections 3.2.1 and 3.2.2 describe geometric monitoring (GM) and apply it to the entropy problem. Entropy monitoring, however, poses a unique challenge when using GM, which we address in Section 3.2.3. We also provide an entropy sketch version.
Algorithm 3.1: Node j when events enter or exit the window.

1. increase (or decrease) counts \( n^j, b_j^i \)
2. update local vector: \( P^j \leftarrow \frac{P^j}{n^j} \)
3. re-weight slack: \( S^j \leftarrow \frac{n^j}{n^0} S_0^j \)
4. if \( P^j - S^j \notin C_L \) or \( P^j - S^j \notin C_U \) then
   5. report safe zone violation and \( P^j, n^j \) to coordinator
   6. wait for response with updated \( h_0, C_L, C_H, S_0^j, \) and/or \( n_0^j \)

Algorithm 3.2: Coordinator violation resolution (eager sync).

1. poll nodes for their \( P^j, n_j \)
2. compute updated \( h_0, C_L, C_H, S_0^j, n_0^j \) and send to nodes

for high dimensional problems to reduce both bandwidth and memory (Section 3.2.4). To support estimating entropy of time-based windows, Section 3.2.5 extends the GM formulation to this setting using a new slack re-weighting scheme. Section 3.2.6 describes an alternative to sync on all violation, extends it to the new time-based formulation, and proposes a new balancing strategy. Finally, we briefly discuss estimating generalizations of entropy (Section 3.2.7).

The resulting CIDER estimator is detailed in Alg. 3.1, Alg. 3.2 and Alg. 3.3.

3.2.1 Basic Geometric Monitoring

Let \( f \) be the function we wish to monitor for threshold crossings. In our case, \( f(P) = H_{MLE}(P) = -\sum^{k}_{i=1} p_i \ln p_i \). Let \( P_0 \) be the reference point, the value of \( P \) during last sync, and similarly \( P_0^j \) denotes the value of \( P^j \) during last sync. Finally, define weights \( \alpha^j = \frac{n^j}{n} \) and the weights during last sync \( \alpha^0_0 = \frac{n_0^j}{n_0} \). We initially assume that each sliding window has a fixed, constant size: \( n^j = n^0_j, n = n_0 \) and \( \alpha^0_0 = \alpha^j \). Section 3.2.5 extends the formulation to time-based sliding windows where \( n^j \)’s change over time and global \( n \) is unknown.

Applying geometric monitoring can be distilled to two steps. In the first, we show how to compute \( f(P) \) from the weighted mean of local vectors \( P^j \) at each node. In our case, the global probability vector \( P = \sum^{m}_{j=1} \alpha^j P^j \) since \( p_i = \frac{1}{n} \sum_j b^j_i = \frac{1}{n} \sum_j n^j p_i^j = \sum_j \alpha^j_i p^j_i \).

In the second step we convert each threshold constraint to a convex set of probability vectors called the safe zone, such that if local probabilities lie inside the safe zone, the constraint is satisfied. Setting convex constraints in the function domain is the key insight in geometric monitoring [LGKS17, Wol15]: if all local vectors lie inside a convex set, then their weighted mean (i.e., the global vector) must also lie inside it, and since entropy is a function of this weighted mean, it follows that the threshold constraint is satisfied. Formally, for any convex safe zone \( C \): \( \forall j : P^j \in C \implies P = \sum_j \alpha_j P^j \in C \).

Local constraint violations are possible even when the global aggregate vector is still
inside the safe zone; these spurious local violations may incur extra synchronizations. However, the reverse is impossible: if \( f(P) \) crosses the threshold at any point, at least one local \( f(P_j) \) also crosses it. Since there can be no missed violations of the global thresholds, our approximation bounds are guaranteed.

The following lemma shows how to derive convex safe zones for the two thresholds for any concave function.

**Lemma 3.2.1 (Safe Zones for Concave Approximations).** Let \( P_j \) and \( P = \sum_{j=1}^{m} \alpha_j P_j \) be the current local and global vectors, \( P_0^j \) and \( P_0 \) their values during the last sync, and assume that \( T_L \leq f(P_0) \leq T_U \). Then for a concave function \( f(P) \):

**Lower Threshold:** The optimal convex safe zone for the lower threshold is \( C_L = \{ P_j : f(P_j) \geq T_L \} \), and \( \forall j : P_j \in C_L \implies f(P) \geq T_L \).

**Upper Threshold:** The optimal convex safe zone for the upper threshold is \( C_U = \{ P_j : \langle P_0 - Q, P_j - Q \rangle \geq 0 \} \), where \( \langle \rangle \) denotes the inner product and \( Q \) is the point on the surface \( f(P) = T_U \) closest to \( P_0 \). It guarantees: \( \forall j : P_j \in C_H \implies f(P) \leq T_U \).

**Closest Point:** \( Q \) can be found via convex optimization or with a closed-form solution.

**Proof. Lower Threshold:** We aim to find a convex local constraint \( C_L \) such that \( \forall j : P_j \in C_L \implies f(P) \geq T_L \).

Since \( f \) is concave, the set \( f(P) \geq T_L \) is convex, so we can simply use \( f(P^j) \geq T_L \) as our local constraint:

\[
C_L = \{ P_j : f(P_j) \geq T_L \}.
\]

\( C_L \) is optimal as it contains all points that satisfy the constraint. Since \( C_L \) is convex, it follows that \( \forall j : P_j \in C_L \implies P \in C_L \implies f(P) \geq T_L \).
Upper Threshold: We aim to find a convex local constraint $C_U$ such that $\forall j : P^j \in C_U \implies f(P) \leq T_U$.

Since $f$ is concave, the set $f(P) \leq T_U$ is not convex, but its complement $f(P) > T_U$ is convex. Let $Q$ be the closest point to $P_0$ on the surface $f(P) = T_U$. $P_0 - Q$ is perpendicular to this convex surface; thus it is the normal to the tangent plane to $f$ at $Q$ (its direction is equal to the gradient at $Q$). This tangent plane is the boundary of the optimal convex subset [LSK+15, Theorem 3] so the local constraint is:

$$C_U = \{ P^j : \langle P_0 - Q, P^j - Q \rangle \geq 0 \}.$$

Since $C_U \subset \{ P^j : f(P) \leq T_U \}$ and is convex, $\forall j : P^j \in C_U \implies f(P) \leq T_U$.

Closest Point: $Q$ is the closest point on the surface $f(P) = T_U$ to the reference point $P_0$. Finding the closest point on a convex surface or set is a well-studied problem, and for many functions very efficient and even closed-form solutions are known. Here we limit ourselves to showing that this is a convex optimization problem:

$$Q = \text{argmin}_P \| P - P_0 \|^2 \ \text{s.t.} \ \ T_U - f(P) \leq 0$$

Since $f$ is concave, the above constraint $f(P) \geq T_U$ is convex and guarantees the minimizer $Q$ will have $f(Q) = T_U$. Both objective and constraint are convex, and therefore this is always a convex minimization problem.

The equivalent lemma for convex $f$ is symmetric: the upper safe zone becomes $C_U = \{ P^j : f(P^j) \leq T_U \}$, while the lower safe zone is $C_L = \{ P^j : \langle P_0 - Q, P^j - Q \rangle \geq 0 \}$, where $Q$ is the closest point to $P_0$ on the surface $f(P) = T_L$.

Entropy Approximation

We can finally apply Lemma 3.2.1 to derive local safe zones for $f = H_{\text{MLE}}$. Recall that the entropy function $f(P) = -\sum_{i=1}^{k} p_i \ln p_i$ is a concave function in $P$. We also need to add a second constraint to the optimization problem, $\sum_{i=1}^{k} p_i = 1$, to ensure that the probabilities in $Q$ sum to 1. This is a convex constraint, and so the problem remains a convex minimization problem. Figure 3.1(a) demonstrates the shape of the lower safe zone. Figure 3.1(b) shows the time it takes to find $Q$ with different values of $k$ on an Intel Core i7-4500U CPU running at 1.80GHz, using the CVXPY convex optimization package [DB16].

Binary Search For Tangent Plane

When $k$ is too high or if the coordinator is processing-limited, we replace the convex optimization procedure with a heuristic: denote by $P_{\text{max}}$ the point with highest entropy $[1/k, \ldots, 1/k]$, and use the tangent plane at $\tilde{Q}$, defined as the point where the surface

\footnote{For convex $f$, the first constraint becomes $f(P) - T_L \leq 0$, which is still convex.}
We can avoid unnecessary synchronization by choosing the drift vector $D^j$ to be centered on $0$, meaning $P_0 + D^j$ is close to $P_0$ and hopefully closer to $P$ than to $P^j$. Since $P = \sum_j \alpha^j P^j = P_0 + \sum_j \alpha^j D^j$, nodes can check $P_0 + D^j \in C$ instead of $P^j \in C$, and retain the threshold monitoring guarantee. (In an equivalent formulation, $C$ is shifted to center on $0$, and nodes check $D^j \in C$ directly. This is used in Chapter 4.) Defining slack to be $S^j_0 = P^j_0 - P_0$ gives us exactly $W^j = P^j - S^j_0 = P_0 + D^j$. Figure 3.2 illustrates

$$f(P) = T_U$$ intersects with the segment $P_0 P_{\max}$. Since $f$ is concave and $P_{\max}$ is a global maximum, $f$’s value on the segment is monotonic, meaning we can efficiently find $\tilde{Q}$ to arbitrary precision using binary search. The resulting safe zone is not optimal, but in practice it performs well.

### 3.2.2 Additive Slack and Drift

Consider the following example with 2 nodes: $P^1_0 = [1/2, 1/2]$, $P^2_0 = [1, 0]$, and $P_0 = (P^1_0 + P^2_0)/2 = [3/4, 1/4]$. Node 1 has maximum possible entropy $f(P^1_0) = \ln 2$ while node 2 has minimal entropy $f(P^2_0) = 0$. Any non-trivial safe zones we compute from $f(P_0) \approx 0.562$ would be immediately violated. In other words, if $P^j_0$ is too far from $P_0$, synchronization may not resolve local violations. Moreover, it becomes more likely that $P \in C$ while $P^j \not\in C$. We use drift, a form of additive slack, to address these problems.

#### Additive Slack

Recall that the global vector is the weighted mean of the local vectors, $P = \sum_j \alpha^j P^j$, that $\alpha^j_0 = \alpha^j$, and that the safe zone $C$ is convex. Let $S^j_0$ be node $j$’s slack vector, allocated by the coordinator during synchronization such that $\sum_{j=1}^m \alpha^j S^j_0 = 0_k$, where $0_k$ is the zero vector of length $k$. Each node now checks if $W^j = P^j - S^j_0 \in C$ rather than $P^j \in C$. Since $\sum_j \alpha^j W^j = \sum_j \alpha^j P^j - 0_k = P$, and from convexity of the safe zones, it follows that if $W^j \in C_U$ and $W^j \in C_L$ for all nodes, then $T_L \leq f(P) \leq T_U$. We can avoid unnecessary synchronization by choosing $S^j_0$ such that $f(W^j)$ is closer to $f(P)$ than to $f(P^j)$.

#### Drift

The drift vector $D^j = P^j - P^j_0$ is the change in the local vector of node $j$ since the last sync. Changes in the data are expected to be gradual, so drifts are likely small, meaning $P_0 + D^j$ is close to $P_0$ and hopefully closer to $P$ than to $P^j$. Since $P = \sum_j \alpha^j P^j = P_0 + \sum_j \alpha^j D^j$, nodes can check $P_0 + D^j \in C$ instead of $P^j \in C$, and retain the threshold monitoring guarantee. (In an equivalent formulation, $C$ is shifted to center on $0$, and nodes check $D^j \in C$ directly. This is used in Chapter 4.) Defining slack to be $S^j_0 = P^j_0 - P_0$ gives us exactly $W^j = P^j - S^j_0 = P_0 + D^j$. Figure 3.2 illustrates

![Figure 3.2: Global, local, drift, and slack vectors with two nodes. The local vectors are outside the safe zone (shaded gray), while $W^1$, $W^2$ and $P = \frac{W^1 + W^2}{2}$ are inside the safe zone.](image-url)
drift slack with two nodes. We use drift slack for both the lower and upper threshold monitors.

Returning to the example, after sync we have $S_0^1 = [-1/4, 1/4]$ and $S_0^2 = [1/4, -1/4]$, thus $P_0^1 - S_0^1 = P_0^2 - S_0^2 = P_0 \in \mathcal{C}$.

### 3.2.3 Extending the Domain of $H_{MLE}$

One complication unique to entropy is that elements of $P$ are assumed to be probabilities: $p_i \geq 0$ and $\sum_{i=1}^k p_i = 1$. Entropy $f(P) = -\sum_{i=1}^k p_i \ln p_i$ is undefined if any $p_i < 0$. Subtracting the slack vectors from the probabilities may cause them to fall outside $f$’s domain, making it impossible to compute $f(W^j)$ when checking whether $W^j \in \mathcal{C}_L$. One simple solution is for nodes to declare a safe zone violation and force a synchronization if any element of $W^j$ is above 1 or below 0, but this may result in many spurious violations since $P^j$ changes while $S_0^j$ was fixed during the last sync.

We address this in two ways. First, we allocate slack vectors $S_0^j$ such that their elements sum to zero so $\sum_{i=1}^k W^j_i = 1$. For drift slack, this happens naturally. Second, we “extend” the domain of $f$ by defining another function $\tilde{f}$ such that: (a) $\tilde{f}$ is defined everywhere (its domain is $\mathbb{R}^k$); (b) $\forall P \in \mathbb{R}_{+}^k : \tilde{f}(P) \leq f(P)$; and (c) $\tilde{f}$ is concave.

Unfortunately, it is impossible to extend $f$’s domain to the negative range and maintain concavity: $\lim_{x \to 0^+} g'(x) = \infty$, where $g(x) = -x \ln x$. Instead, we define $\tilde{f}(P) = \sum_{i=1}^k \tilde{g}(p_i)$, where:

$$
\tilde{g}(x) = \begin{cases} 
-x \ln x & \text{if } x \geq x_0 \\
-x \ln x_0 & \text{if } x < x_0 
\end{cases}
$$

and $x_0 \in (0, 1]$ is a suitable small constant. Note that $\tilde{f}$ meets the above criteria since $\tilde{g}$ is defined everywhere, $\tilde{g}(x) \leq g(x)$ for all $x > 0$, and $\tilde{g}(x)$ is concave\(^6\).

Figure 3.3 shows $g(x)$ and $\tilde{g}(x)$. There is a tradeoff in setting $x_0$: if $x_0$ is too high, then $\tilde{f}$ is too far below $f$ meaning that $f(W^j) < T_L$ more often. Yet setting $x_0$ too low makes $\ln x_0$ large, so negative elements of $W^j$ could overwhelm the positive ones. In practice, our experience shows that negative elements in $W^j$ are rare and tend to be

\(^6\)It is continuous and $\forall a, b \in \mathbb{R}, \alpha \in [0, 1] : \tilde{g}(\alpha a + (1 - \alpha)b) \geq \alpha \tilde{g}(a) + (1 - \alpha)\tilde{g}(b)$.\n
---

*Figure 3.3: $g(x) = -x \ln x$ and its concave “extension” to the negative domain, $\tilde{g}(x)$. $x_0$ is set to 0.2 for illustration.*
much smaller than the positive ones, so it is safe to set $x_0$ to a very small number. We propose the heuristic $x_0 \approx \frac{1}{100}$.

Having defined concave $\hat{f}$, we can monitor the lower bound using geometric monitoring: the new convex lower safe zone is $\tilde{C}_L = \{ W^j : \hat{f}(W^j) \geq T_L \}$. If $\hat{f}(W^j) \geq T_L$ for all nodes, then $f(P) \geq \hat{f}(P) \geq T_L$. Each node simply checks whether $W^j \in \tilde{C}_L$.

### 3.2.4 Entropy Sketch

What happens when $k$, the number of bins or event types, is too large? For example, for network monitoring we estimate the entropy of IP addresses [BJS15], where $k = 2^{32}$ (for IPv4), or even $k = 2^{64}$ (origin-destination pairs [ZLO+07]). It is infeasible to have so many counters in memory or send messages with local vectors of such size. In such cases we use sketching to estimate entropy.

An entropy sketch $\hat{H}$ is a data structure $Y$ of size $s$ and corresponding functions $h(Y)$ such that $s \ll k$ and $H_{MLE} \approx h(Y)$ with probabilistic error bounds that usually depend on the relative size of $s$ and $k$. Incoming and outgoing observations in the stream update $Y$ instead of $P$. Sketching is also effective in reducing bandwidth, since $s \ll k$, and is often proposed as a distributed entropy approximation [ZLO+07, HNO08].

We describe a CIDER variant that allows high $k$ and bandwidth reduction by adopting Clifford and Cosma’s near-optimal entropy sketch [CC13]. This sketch is surprisingly simple: $Y$ is a linear projection of $P$, and $h(Y)$ is a concave function. Denote by $B$ the vector of global counts $[b_1, b_2, \ldots, b_k]$. Then $Y = \frac{R \cdot B}{n} = R \cdot P$ where $R$ is a $s \times k$ random projection matrix with i.i.d elements drawn from $F(x; 1, -1, \pi/2, 0)$ and $h(Y) = -\ln \left( \frac{1}{k} \sum_{i=1}^k \exp(y_i) \right)$. Since the sketch is linear, the global sketch vector $Y = R \cdot P$ is the weighted mean of the local sketch vectors $Y^j = R \cdot P^j$:

$$Y = R \cdot P = R \sum_{j=1}^m \alpha^j P^j = \sum_{j=1}^m \alpha^j R \cdot P^j = \sum_{j=1}^m \alpha^j Y^j$$

$Y^j$ and $Y$ completely replace $P^j$ and $P$ in the algorithm: nodes only maintain and communicate sketches of size $s$, and need not maintain $k$ explicit counters in memory. We monitor $\hat{H}$ in the same way that we monitor $H_{MLE}$: we generate a fixed $R$ (using the procedure in [CC13, Table 1]), apply the sketch to obtain local vectors $Y^j$ ([CC13, Table 2] offers an equivalent incremental formulation of the sketch), and use local safe zones for the lower and upper bound to guarantee the approximation. Since $h(Y)$ is a function of the weighted mean of local vectors, and since it is concave, we can directly apply Lemma 3.2.1 to derive local safe zones. Since $Y^j$ are not probabilities, we need not constrain $Q$’s elements to sum to 1. Moreover, since $h$ is defined everywhere, we do not use the full domain extension trick from Section 3.2.3. The resulting approximation is probabilistic rather than deterministic, since we are estimating $\hat{H}$ and not $H_{MLE}$ (see [CC13] for details).

---

7The function $\ln \left( \sum_i \exp(y_i) \right)$ is known as Log-Sum-Exp and is convex [CE14].
3.2.5 Dynamically-sized Sliding Windows

So far we have assumed fixed window size: \( n^j \) is constant, and \( n^j = n^j_0 \). Yet in many entropy monitoring settings (e.g., network monitoring [GGSZ14]) observations can arrive to different nodes at different times and rates, so that the same node can have (sometimes vastly!) different numbers of observations in the window. Though geometric monitoring has been previously used to build communication-efficient approximations [GKS15, LKS16, GKS14], previous work has always assumed fixed-size windows with identical sizes \( n^j = \frac{n_m}{m} \), and that events arrive to all streams at a constant, identical rate. To support true time-based sliding windows, we must extend geometric monitoring to support variable-sized windows using a new slack re-weighting scheme. This new scheme is applicable to most geometric monitoring applications.

We first observe that the original convexity argument holds even if nodes do not actually know the values of \( \alpha^j \). Since \( n^j \), the current number of observations in node \( j \)'s sliding window, changes dynamically, the global \( n \) is unknown to any node, even the coordinator. Nevertheless, by definition, \( \alpha^j \) are still positive and sum to 1, so \( P = \sum_{j=1}^{m} \alpha^j P^j \). As before \( \forall j : P^j \in C \implies P \in C \). This applies to both entropy and entropy sketch.

### Slack Re-weighting

One problem with variable-size windows is that previous additive slack schemes, including drift, no longer work correctly, because they assume that the number of observations at each node is fixed. Since we now allow \( n^j \) to change, the current weights \( \alpha^j \) do not match the original slack weights \( \alpha^j_0 \).

Denote by \( \Box_0 \) the value of quantity \( \Box \) during last sync and recall that:

\[
P^j = \frac{B^j}{n^j} \quad , \quad P^j = \frac{B^j_0}{n^j_0} \quad , \quad \alpha^j = \frac{n^j}{n} \quad , \quad \alpha^j_0 = \frac{n^j_0}{n_0}
\]

\[
P = \frac{n^j}{n} P^j = \sum_j \alpha^j P^j \quad , \quad P_0 = \sum_j \frac{n^j_0}{n_0} P^j_0 = \sum_j \alpha^j_0 P^j_0.
\]

For drift slack, \( S^j_0 = P^j_0 - P^j \), and from Section 3.2.2 we know that

\[
\sum_j \alpha^j_0 S^j_0 = \sum_j \alpha^j_0 P^j_0 - \alpha^j_0 P_0 = \sum_j \left( \alpha^j_0 P^j_0 \right) - P_0
\]

\[
= \sum_j \left( \frac{n^j_0 B^j}{n_0 n^j_0} \right) - P_0 = \sum_j \left( \frac{B^j_0}{n_0} \right) - P_0
\]

\[
= P_0 - P_0 = 0_m.
\]

Yet applying the original additive slacks \( S^j_0 \) to the current \( P^j \) will not preserve \( P \), even
though \( \sum_j \alpha_j^i S_j^0 = 0_m \), since in general \( \alpha^j \neq \alpha_0^j \):

\[
\sum_j \alpha^j (P^j - S^j) = \sum_j \left( \alpha^j P^j - \alpha^j P_0^j + \alpha^j P_0 \right)
= P - \sum_j \alpha^j P_0^j + \sum_j \alpha^j P_0
\neq P.
\]

We address this problem using a novel slack re-weighting scheme: \( S^j = \frac{n_j^j}{n^j} S_0^j \).

Though the global \( n \) is unknown, nodes know the values of their local \( n_j^j \) and \( n_j^0 \) so each can individually update its slack. With the new scheme:

\[
\sum_j \alpha^j (P^j - S^j) = \sum_j \left( \alpha^j P^j - \alpha^j n_j^0 \frac{n_j^j}{n^j} S_0^j \right)
= P - \sum \frac{n_j^j}{n} \frac{n_j^0}{n} S_0^j
= P - \frac{n_0}{n} \sum \alpha_j^0 S_0^j = P - \frac{n_0}{n} 0_m = P.
\]

It follows that for a convex safe zone \( \mathcal{C} \), if for all nodes \( W^j = P^j - S^j \in \mathcal{C} \), then \( P \in \mathcal{C} \).

This slack re-weighting applies to any valid additive slack, as long as \( \sum_j \alpha_0^j S_0^j = 0_m \).

Moreover, it does not depend on the function being monitored and can be applied to other geometric monitoring algorithms.

### 3.2.6 Violation Resolution

When a node reports safe zone violation, the coordinator must resolve it. The simplest policy is full or eager synchronization [GKS13]: poll all nodes for their local vectors and recompute \( P_0 \), approximation \( h_0 = f(P_0) \), thresholds \( T_L = LB(h_0), T_U = UB(h_0) \), safe zones \( \mathcal{C}_L, \mathcal{C}_U \), local counts \( n_j^0 \), and slack vectors \( S_0^j \); updated values are sent to nodes and monitoring resumes. Eager sync is costly, however, and can be wasteful since many local safe zone violations occur when the current approximation is still within bounds (as seen, for example, in Figure 2.5(c) in Section 2.4).

An alternative policy is lazy synchronization [GKS13]: poll the other nodes one at a time, and add them to a balancing set \( \mathcal{S} \) until either the weighted mean of their local vectors \( P^V = \left(1/\sum_{j \in \mathcal{S}} n_j^j \right) \sum_{j \in \mathcal{S}} n_j^j P^j \) is inside the safe zones, or an upper limit \( \tilde{m} < m \) on the size of \( \mathcal{S} \) has been reached. If the set of nodes with violations can indeed be balanced, the coordinator updates \( S_0^j \) and \( n_0^j \) for the nodes in \( \mathcal{S} \) (making sure the weighted sum is unchanged), all without contacting or sending updates to nodes not in \( \mathcal{S} \). If \( |\mathcal{S}| \) is too large, we assume that \( P \) itself is outside the safe zone and trigger an eager sync. Lazy sync with dynamically-sized sliding windows requires careful re-weighting of slack; Alg. 3.3 contains the full details.

Given the set of violating nodes, which node should the coordinator poll next? In
Algorithm 3.3: Lazy synchronization with slack re-weighting.

Input: $S$ the set of violating nodes, $\tilde{m}$ max nodes to lazy sync

1. while $|S| < \tilde{m}$ do

2. $n^V, n_0^V \leftarrow \sum_{j \in S} n^V_j, \sum_{j \in S} n_0^V_j$

3. $P^V, S^V \leftarrow \sum_{j \in S} \frac{n^V_j}{n_0^V} P^j, \sum_{j \in S} \frac{n^V_j}{n_0^V} S_0^j$

4. if $P^V - S^V \in C_L$ and $P^V - S^V \in C_U$ then break

5. pick node $j \notin S$ to poll with balance strategy, add $j$ to $S$

6. if $|S| = \tilde{m}$ then switch to eager synchronization (Alg. 3.2)

7. forall $j \in S$ do

8. send update to $j$: $n_0^j, S_0^j \leftarrow \frac{n_0^V}{n_0^V} n_0^j, \frac{n_0^V}{n_0^V} n_0^j \leftarrow \left( P_j - (P^V - S^V) \right)$

random balancing, the coordinator simply chooses a random node. Previous work has proposed the least-used (LU) strategy: choose the non-violating node that has been polled least recently [GKS13].

We propose a new balancing strategy we call opposite slack (“oslack”): the coordinator polls the non-violating node $j$ whose slack $S_0^j$ has the opposite direction to the weighted mean of local slacks of the nodes combined so far ($S^V$ in Alg. 3.3). The coordinator knows the values of $S_0^j$ for all nodes, so no extra communication is incurred. The intuition behind oslack is that since the current safe zone is “centered” around $P_0$, we want to poll a node $j$ such that $P_j - P_0$ balances $P^V - P_0$. If $V$ “pulls” $P$ in one direction, we want to find a node that “pulls” it back into the safe zone. $P_j$ is unknown for unpolled nodes, however. The idea behind oslack is that slack $S_0^j$ can serve as a proxy for $P_j - P_0$. Drift slack is $P_0^j - P_0$, and if local vectors have not shifted by much, $S_0^j$ is close to $P_j - P_0$. Figure 3.2 illustrates this intuition: though $P_1^1 \neq P_1^0$ and $P_2^2 \neq P_2^0$, the slacks $S_0^1$ and $S_0^2$ still give a rough idea of the directions $P_1^1 - P_0$ and $P_2^2 - P_0$.

3.2.7 Approximating Rényi and Tsallis Entropies

We briefly discuss approximating two parametric generalizations of entropy: the Tsallis entropy

$$H_\beta = \frac{1}{1-\beta} \left( \sum_{i=1}^k \Pr[X_i]^{\beta} - 1 \right) ,$$

and the Rényi entropy

$$H_\alpha = \frac{1}{1-\alpha} \ln \left( \sum_{i=1}^k \Pr[X_i]^\alpha \right) .$$

The Tsallis entropy is convex for $\beta < 0$ and concave for $\beta > 0$ [HV15], and can be computed from the weighted mean of local probabilities $P_j$. Therefore, we can apply Lemma 3.2.1 to derive safe zones and monitor it the same way we do for Shannon
entropy and the sketch. $H_\beta$ is defined everywhere, so we can safely use slack.

The Rényi entropy is concave for $\alpha \in (0, 1]$ (so Lemma 3.2.1 applies directly) and quasiconcave for all $\alpha \geq 0$ [HV15]. Quasiconcave functions have convex upper level sets, so the lower bound constraint $\{P | H_\alpha(P) \geq T_L\}$ is convex, and the upper constraint $\{P | H_\alpha(P) \leq T_H\}$ is “reverse convex”. Therefore we can use the same half-plane trick from Lemma 3.2.1 to derive optimal safe zones. The optimization problem of finding the closest point $Q$ now has a quasiconvex constraint. Though quasiconvex optimization problems have been extensively studied, we consider this beyond the scope of this work.

### 3.3 Empirical Evaluation

We evaluate CIDER’s performance on several public datasets representing real-world application scenarios. We simulate running the nodes and coordinator using the recorded timestamps in the dataset, keep track of the estimated and true entropy, and count any messages sent. Unless otherwise noted, we use absolute error bounds $LB(h_0) = h_0 - \epsilon$ and $UB(h_0) = h_0 + \epsilon$ where $\epsilon$ is the desired accuracy.

Our baseline is the centralized estimator where each node sends new observations to the coordinator as they arrive (unlike CIDER, it does not provide the global estimate to the nodes themselves). Our main performance metric is communication fraction, defined as the ratio of messages sent by CIDER to those sent by the centralized algorithm. For both algorithms, we exclude messages sent in the first window.

#### 3.3.1 Network Traffic Monitoring

Traffic entropy (e.g., of IP addresses, ports, or origin-destination flows) is a commonly-suggested feature for network health monitoring [GGSZ14, BJS15, ZLO+07, ABBS10, NSA+08]. Estimating traffic entropy is a thorny problem, however, due to the volume and rate of incoming network packets, distributed input sources (often routers), and the need for timely detection. Existing systems use a combination of periodic reporting (increasing latency), sketching (increasing error), and sub-sampling (increasing both error and risk of missed events). CIDER offers an alternative: low communication, deterministic bounds, and near-realtime detection of changes in entropy.

We evaluate CIDER’s performance on the CTU-13 collection [GGSZ14], a set of publicly available annotated traffic captures. We use the bidirectional NetFlow files from the CTU1, CTU4, CTU9 and CTU10 datasets, each containing 1.1M–2.8M NetFlows recorded over 4–6 hours. NetFlows are assigned to simulated “routers” (nodes) by uniformly dividing the 3rd octet of the destination IP to $m$ bins, and we approximate the entropy of source IP address ($k = 2^{32}$) with sketch of size $s = 100$. Unless otherwise noted, we simulate $m = 10$ nodes with five minute sliding windows, use lazy synchronization with up to $\tilde{m} = 3$ nodes and oslack balancing, and require accuracy $\epsilon = 0.5$. Section 3.3.4 uses CTU2 to select $\tilde{m}$ and balance strategy.
Figure 3.4: Entropy of source IP addresses in the CTU1 dataset, with 10 nodes and a five minute sliding window (top) and CIDER approximation error (black, bottom) with accuracy $\epsilon = 0.5$ (dashed horizontal line). CIDER maintains the desired accuracy with 1.2% communication. An equivalent periodic estimator (green) is unable to maintain this accuracy.

Figure 3.4 shows an example of such a simulation on CTU1 (length 6 hours and 15 minutes). CIDER maintains accurate approximation using only 1.2% communication of the centralized algorithm – equivalent to nodes sending updates every 4 minutes and 30 seconds. Such a periodic estimator would be unable to maintain the desired accuracy $\epsilon = 0.5$, however: its estimation error reaches almost 3.0. This translates to a four minute delay in detecting any changes, and short entropy spikes can be missed entirely. CIDER’s approximation error, on the other hand, is always within bounds and guarantees no detection latency or missed spikes.

Accuracy $\epsilon$

Figure 3.5 shows CIDER’s communication with different approximation accuracies on the CTU1, CTU4, CTU9, and CTU10 datasets. Lazy synchronization with oslack is considerably more communication-efficient than eager synchronization, with up to an order of magnitude difference. In general, communication is reduced by roughly two orders of magnitude, even with a fairly strict $\epsilon = 0.5$. Even with highest accuracy $\epsilon = 0.1, 0.5\%$ of the total entropy range $0–\ln 2^{32}$, CIDER only sends 10% messages as the centralized algorithm. On the other hand, with a relaxed $\epsilon = 2.0$, enough to catch the large entropy spikes in CTU1 (Figure 3.4), CIDER yields 0.7% communication.

Window Size

CIDER’s performance does not inherently depend on the window size, and instead reflects the behavior of local and global probabilities. Figure 3.6 illustrates this on the
**Figure 3.5:** Communication for CIDER at different approximation accuracies when monitoring source IP entropy in the CTU datasets.

**Figure 3.6:** Larger sliding windows smooth out variations in probabilities and therefore entropy (left), resulting in lower communication when monitoring the CTU1 dataset (right).
CTU1 dataset. Increasing window size means more observations, which in turn means that probabilities change gradually rather than abruptly. As entropy spikes become smaller and even disappear, CIDER adapts and requires less communication to maintain the same accuracy.

**Number of Nodes $m$**

Figure 3.7 explores how different violation resolution strategies scale as we increase the number of simulated nodes $m$ in CTU1. Eager synchronization scales poorly, and with 50 nodes it performs no better than the centralized algorithm. Lazy sync with LU or random balancing can scale up to 70 nodes while still reducing communication. The oslack policy, however, scales much better and can monitor even 100 nodes with only 30% communication. Section 3.3.4 describes how we set lazy sync parameters.

### 3.3.2 Load Balancing

In the classic load balancing scenario, work items from an input stream must be assigned to one of $k$ workers (e.g., based on a hash function). Entropy of worker load distribution can be used to identify load imbalance, and react accordingly (add workers, change work distribution, etc.). Consider the case where there are $m$ distributed streams. Though each node (“balancer”) has a local count of worker assignments, it cannot know whether global workload is balanced. Similar situations arise in key-value stores, distributed stream processing systems [NDFMGS+15], and more. We observe that in this settings sketching approaches cannot help, since $k$ is typically low. CIDER provides a view on global entropy to every node, allowing them to adjust work allocation as needed.

The WorldCup’98 dataset [AJ98] contains access logs for the 1998 World Cup web site for a 3 months period. We define the WC98 dataset as the access logs from June 10, 1998 (day 46, with 50M requests), where server ID for each request is used as the
node \((m = 27)\), and the request type (image, html, etc., \(k = 13\)) serves as the assigned worker (the observation). For this dataset we use \(\varepsilon = 0.05\), 10 minute sliding windows, and \(\max n = 5\) in lazy sync. We used data from day 44 to tune these parameters (Section 3.3.4).

Figure 3.8 shows CIDER’s performance on the WC98 dataset for accuracies in the very tight range 0.005–0.05 (which is 0.2%–2% of maximum entropy, \(\ln 13 \approx 2.565\)). In this range, CIDER reduces communication by between one and two orders of magnitude. Lazy sync is even more critical here, with an almost 10 times communication reduction compared to eager synchronization. In practice we might only care about large changes in entropy. With \(\varepsilon = 0.25\) (10% of entropy range), for example, CIDER almost never syncs, requiring only 0.25% communication. This illustrates one advantage of CIDER over existing approaches: when entropy changes slowly, practically no communication is necessary; if entropy starts to change rapidly, CIDER will ramp up communications to adapt.

3.3.3 Air Quality Monitoring

The Revised Air Quality Index [CKL+04] (RAQI) combines metrics from five air pollutants\(^8\) into a single number, and incorporates Shannon entropy as one of its components to represent variations within the measurement period. This component is the entropy over a window of 24 hours of \(\max\{A_1 \ldots A_5\}\), where \(A_i\) are individual pollutant measurements converted to AQI scores using the piecewise linear transformation described in [AQI16].

We used pollutant data from North Taiwan available from the Taiwan Air Quality Monitoring Protection Agency\(^9\) to track this entropy component. The TAQ dataset is defined as the hourly data from January 1 through July 31 2015 for the \(m = 23\)

---

\(^8\)PM\(_{10}\), O\(_3\), SO\(_2\), CO, and NO\(_2\)

monitoring stations that report the five pollutants. The first two months are used to tune \( \tilde{m} \) for lazy synchronization (Section 3.3.4), and the rest for evaluation. We divided the AQI range of 1–500 to \( k = 20 \) uniform bins. We use lazy sync with oslack balancing, with \( m = 11 \) max nodes, and the standard RAQI sliding window of 24 hours.

Performance of CIDER on the TAQ dataset is shown in Figure 3.9. Entropy fluctuates widely, both due to large hourly differences, but also because local windows only have \( n^j = 24 \) observations in the window so the MLE estimator is at the lower range of accuracy (where \( n^j \sim k \) [Pan03, NBdRvS04]). CIDER must sync often (as would any other algorithm), but is still able to reduce communication to 17%–85%. For example, in remote, battery-operated wireless monitoring stations, this would translate to substantial increases in battery life.

3.3.4 Lazy Synchronization

Lazy synchronization is superior to eager synchronization, but we must still set \( \tilde{m} \), the maximum size of the balancing set, and select a balancing strategy (random, LU, or oslack).

Figure 3.10 shows CIDER’s performance with different values of \( \tilde{m} \), for each of the three datasets. To avoid overfitting, we used different data to tune \( \tilde{m} \). LU and random achieve similar communication on all datasets, and are essentially equivalent. On the WC98 and TAQ datasets, oslack performance is equivalent to the other two, especially at the optimal value of \( \tilde{m} \). On the CTU2 dataset, however, oslack is substantially better, especially with 70 simulated nodes. We conclude that oslack performs as well or better than other balancing strategies, and use it throughout. This is further confirmed by Figure 3.7 where LU performance is identical to that of random, while oslack balancing is essential to good scaling.
Figure 3.10: Communication vs. max number of balancing nodes on different datasets. On CTU2, random balancing works as well as LU, while oslack performs considerably better than both. On the other datasets all three perform similarly at the optimal \( \bar{m} \).
Chapter 4

Monitoring Least Squares Models of Distributed Streams

Least squares regression is commonly used for prediction of new values from past values (i.e., forecasting), for analysis of existing phenomena through discovered coefficients (e.g., in econometrics [Hay00] and social studies [RGH+14]), and as building blocks in other algorithms (e.g., in sparse coding [TG07, AEB06]).

Data behavior evolves, however, and changes can render a previously-computed model invalid. In such settings regression models must be updated to incorporate new observations, or to be periodically recomputed. This problem is exacerbated in distributed settings: when observations are distributed over many nodes, we are also faced with the additional cost of communicating updates.

The question then becomes not just how to efficiently compute the model, but when. Recomputing the model after each new observation seems wasteful, as models tend to change slowly. Recomputing periodically still involves needless work if the model changes infrequently, yet may introduce unacceptable errors between scheduled updates.

Hence, there are two complementary approaches to distributed linear regression. The first is efficient distributed computation of the model. Indeed, much work has been devoted to this approach [MBG10, LS06, Say14].

We focus on the second approach – monitoring the quality of a given model, and recomputing it only as needed (using any of the computational approaches). The monitoring approach looks at incoming data and triggers an alert if the previously-learned model is too dissimilar to the hypothetical global model that would have been built given the current data. This problem is difficult in distributed settings, since the existing model and the (hypothetical) current model are both global models – composed from the union of data at all nodes. Thus a distributed monitoring algorithm must deal with communication efficiency, in addition to the problem of monitoring a model without actually relearning it.

The monitoring approach has received little attention, possibly because the least squares solutions involve matrix inversion, which is difficult to analyze (Section 1.3.5).
The few existing techniques monitor the model’s prediction error or $R^2$ fit [BK08, BDG11], or a univariate model where the problem reduces to monitoring a ratio [GRM13]. This is not always sufficient: in some settings we are interested in the model’s coefficients (e.g., analysis), and in others (e.g., interpolation) we lack the ground truth to measure prediction error. Moreover, monitoring model error is a more general approach: prediction error and fit can be inferred from model error but not vice versa.

This Chapter

We describe DILSQ: a novel communication-efficient monitoring algorithm for multivariate least squares models of distributed, dynamic data streams. To our knowledge, this is the first algorithm that monitors the multivariate regression model itself, rather than its prediction or fit. Given a previously-computed global model, we derive local constraints on the local data at each node. A node only communicates if its constraint is broken. These constraints guarantee that if no node communicates, the global hypothetical model is sufficiently close to the precomputed model.

DILSQ easily generalizes to more complex least squares variants such as GLS and RLS [Hay00], and is independent of how the model was computed. Evaluation on two real datasets shows it reduces communication by up to two orders of magnitude. Complete elimination of all communication is also possible when the model is fixed.

4.1 Problem Definition

Let $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a set of $n$ observation pairs of $m < n$ independent variables and one dependent variable, where $x_i$ are column vectors in $\mathbb{R}^m$, and $y_i$ are the corresponding response scalars. We seek a linear transformation $\beta \in \mathbb{R}^m$, $\beta = (\beta_1, \ldots, \beta_m)^T$, that minimizes the sum of squared errors between $y_i$ to the mapping of $x_i$. In other words, we seek a model $\beta$ that minimizes $\|X\beta - y\|^2$, where $X$ is the $n \times m$ matrix of row vectors $X \triangleq (x_1^T, \ldots, x_n^T)^T$, and $y$ is the column vector composed of response scalars $y \triangleq (y_1, \ldots, y_n)^T$.

The optimal solution to this convex problem, known as ordinary least squares (OLS), is given by [Hay00]

$$\beta = (X^T X)^{-1} X^T y.$$  \hspace{1cm} (4.1)

4.1.1 Monitoring OLS of Distributed Streams

Assume that the observations $\{(x_i, y_i)\}$ are distributed across $k$ nodes, and that these observations are dynamic – they change over time, as nodes receive new observations that replace older ones. As data evolves, it is possible that the previously computed model no longer matches the current true model. We wish to maintain an accurate estimation $\beta_0$ of the current global OLS model, $\beta$. The question is then when to update the model.
The simplest way is to update $\beta$ every time a new observation arrives at the nodes, using a straightforward or incremental procedure. Though this gives the most accurate model, it is also wasteful. It requires communicating the update every time, and potentially disseminating the updated model to all nodes. It is especially wasteful when the current global model is similar to the old one.

Another simple solution is the periodic algorithm: sending updates once every $T$ times [WBK09, BK08] guarantees a reduction in communication. The problem is that a fixed update schedule must balance communication and error \textit{a priori}. For large $T$ the estimate error may be unbounded for a long interval, yet if model changes are infrequent we waste communication.

Recent approaches monitor the prediction error $|y - X\beta_0|$, where $X, y$ are the current observations [SWGH13, BK08], the model’s $R^2$ fit [BDG11], or prediction error between divergent local models and the hypothetical global model [KBK$^+14$].

Monitoring prediction error is not always sufficient, however. First, prediction is not the only application of regression. In some settings [Hay00, RGH$^+14$] we are interested in model coefficients, rather than prediction performance. Yet prediction error may be small even when the difference between models is large. Consider the following example in $m = 3$ dimensions, with the precomputed model $\beta_0 = (1, 2, 3)^T$, the current model $\beta = (1, 1, 1)^T$, and with the observation $x = (-0.95, 2.05, -0.95)^T, y = \beta^T x = 0.15$. In this case the prediction error is small, $|x^T \beta_0 - y| = 0.15$, yet the models are very different: $\|\beta_0 - \beta\| = 2.236$.

Monitoring model fit is also tricky. Figure 4.1(a) shows the $R^2$ fit of the true model $\beta$ in an interpolation problem (described in Section 4.3.2). The fit of the true model varies widely, and it is not clear where to set the $R^2$ monitoring threshold. Figure 4.1(b) shows an example where both the model error $\|\beta - \beta_0\|$ and the fit of the monitored model $\beta_0$ are increasing.

Thus, we aim to monitor the model estimation error itself. Let $\beta_0$ be the existing model, previously computed at some point in the past (the synchronization time),
Figure 4.2: Monitoring distributed OLS models is difficult. Current local models $\beta^1, \beta^2$ are identical to the precomputed models $\beta_0^1, \beta_0^2$ but the combined global model is very different, $\beta - \beta_0 = 0.44$.

and let $\beta$ be the hypothetical OLS model from current observations\(^1\). Given an error threshold $\epsilon$, our goal is to raise an alert if

$$||\beta_0 - \beta|| > \epsilon,$$

while minimizing communication. Note that monitoring model error is a more general approach: limiting model error allows us to bound prediction error $|x^T \beta_0 - x^T \beta|$ through Cauchy-Schwarz but not vice versa. Indeed, Sayed and Lopez [LS06] estimate the expected model error and use it to get expected prediction error.

4.2 Monitoring Distributed Least Squares With Convex Subsets

Monitoring distributed OLS models is difficult because the global model cannot be inferred from the local model at each node. Even when all current local models $\beta^j$ are similar to the precomputed local models $\beta_0^j$, the current global model $\beta$ may be very different from the precomputed model $\beta_0$. Consider the example in Figure 4.2 with $k = 2$ nodes and $m = 1$. The global model deviation is very large, $\beta - \beta_0 = 0.44$, even though local models are identical: $\beta^1 = \beta_0^1$ and $\beta^2 = \beta_0^2$.

To overcome this difficulty, we turn to geometric monitoring. Geometric monitoring [KSA+14, KSSL12] is a communication-efficient approach that monitors whether a function of distributed data streams crosses a threshold. The key idea is to impose constraints on local data at the nodes, rather than on the function of the global aggregate. Given a function of the average of all local data and the threshold, we compute a convex safe zone for each node. As we show below, convexity plays a key role in the correctness of this scheme. As long as local data stay inside the safe zones, we guarantee that the function of the global average does not cross a threshold. Nodes communicate only when local data drifts outside the safe zone, which we call a safe zone violation. Once that happens, violations can be resolved, for example by gathering data from all nodes

\(^1\beta$ is hypothetical since we don’t actually compute it.
and recomputing $\beta_0$ and the safe zones.

To summarize, we want to impose conditions on the local data at each node so that as long as they hold, $\|\beta - \beta_0\| \leq \epsilon$. The conditions should be “lenient” as possible – we wish to minimize the number of violations.

### 4.2.1 Notation

Define $A \triangleq \sum_{i=1}^n x_i x_i^T = X^T X$ and $c \triangleq \sum_{i=1}^n x_i y_i = X^T y$, and rewrite Eq. (4.1) as $\beta = A^{-1}c$. The global matrix $A$ can be written as the sum of local matrices $A = \sum_{j=1}^k A^j$, where $A^j$ is constructed from the local observations at node $j$. Similarly, $c = \sum_{j=1}^k c^j$ where $c^j$ is constructed from the local observations at node $j$. Therefore, we can rewrite Eq. (4.1) as a function of the sums of $A^j, c^j$:

$$\beta = \left( \sum_j A^j \right)^{-1} \left( \sum_j c^j \right) = A^{-1}c \quad (4.2)$$

In our notation we use $\{A^j, c^j\}_k$ instead of the original observations $\{x_i, y_i\}_n$. Let $A_0 = \sum_{j=1}^k A^j_0$ and $c_0 = \sum_{j=1}^k c^j_0$ be the global sums of local values at nodes during the last sync time (when $\beta_0$ was computed), and $A = \sum_{j=1}^k A^j, c = \sum_{j=1}^k c^j$ be the current values. We define the local drifts as the deviation of local data from its initial values during sync: $\Delta^j = A^j - A^j_0$ and $\delta^j = c^j - c^j_0$.

We can now express global $\beta$ and $\beta_0$ as a function of the averages of $A^j, c^j$ and $A^j_0, c^j_0$. This will allow us to bound model changes inside a convex subset. Recall $\beta = A^{-1}c$. Similarly, $\beta_0 = A_0^{-1}c_0$. Values averaged over nodes (rather than summed) shall be denoted with $\hat{\cdot}$. Hence initial values

$$\hat{A}_0 = \frac{1}{k} \sum_{j=1}^k A^j_0, \quad \hat{c}_0 = \frac{1}{k} \sum_{j=1}^k c^j_0, \quad \hat{\beta}_0 = \hat{A}_0^{-1} \hat{c}_0,$$

and current values

$$\hat{A} = \frac{1}{k} \sum_{j=1}^k A^j, \quad \hat{c} = \frac{1}{k} \sum_{j=1}^k c^j, \quad \hat{\beta} = \hat{A}^{-1} \hat{c}.$$ 

Note $(\frac{1}{k} A)^{-1} = kA^{-1}$ thus $\hat{\beta} = \hat{A}^{-1} \hat{c} = A^{-1}c = \beta$ and likewise $\hat{\beta}_0 = \beta_0$. In other words, we can compute the OLS model from the averages of local $A^j, c^j$ rather than the sums:

$$\beta = \left( \frac{1}{k} \sum_j A^j \right)^{-1} \left( \frac{1}{k} \sum_j c^j \right) = \hat{A}^{-1} \hat{c} \quad (4.3)$$
4.2.2 Convex Safe Zones

We propose to solve the monitoring problem by means of “good” convex subsets, called safe zones, of the data space. Each node monitors its own drift: as long as current values at local nodes \((A_j, c_j)\) are sufficiently similar to their values at sync time \((A_{0j}, c_{0j})\), \(\beta_0\) is guaranteed to be close to \(\beta\).

Formally, we define a convex subset \(\mathcal{C}\) in the space of matrix-vector pairs, such that 
\[
(0_{m \times m}, 0_m) \in \mathcal{C} \land (\Delta, \delta) \in \mathcal{C} \implies \|\hat{A}_0^{-1}(\hat{c}_0 + \delta) - (\hat{A}_0 + \Delta)^{-1}(\hat{c}_0 + \delta)\| \leq \epsilon ,
\]
(4.4)
for any drift \((\Delta, \delta)\), where \(0_{m \times m}\) and \(0_m\) are the \(m \times m\) zero matrix and length \(m\) zero vector. Ideally, \(\mathcal{C}\) should be “big”: as local data slowly drifts over time, it is desirable that drifts remain in \(\mathcal{C}\) (otherwise communication is needed). Convexity plays a key role in our paradigm: if all drifts are in \(\mathcal{C}\), then their average is also in \(\mathcal{C}\).

Given such a subset \(\mathcal{C}\), the basic monitoring paradigm is simple. As long as \((A_j - A_{0j}, c_j - c_{0j}) \in \mathcal{C}\), node \(j\) can remain silent. If all nodes are silent, then \(\|\hat{\beta}_0 - \hat{\beta}\| = \|\beta_0 - \beta\| \leq \epsilon\). If a violation of the local condition does occur at any node \(j\), some form of violation recovery must take place, for example recomputing the global model and restarting monitoring.

We now prove the correctness of the paradigm.

**Lemma 4.2.1.** Let \(\mathcal{C}\) be a convex subset that satisfies Eq. (4.4). If \((\Delta_j, \delta_j) \in \mathcal{C}\) for all \(j\), then \(\|\beta - \beta_0\| \leq \epsilon\).

**Proof.** Express \(\hat{A}, \hat{c}\) using the average of local deviations:
\[
(\hat{A}, \hat{c}) = \frac{1}{k} \sum_j (A_j, c_j)
\]
\[
= (\hat{A}_0, \hat{c}_0) + \frac{1}{k} \sum_j (A_j - A_{0j}, c_j - c_{0j})
\]
\[
= (\hat{A}_0, \hat{c}_0) + \frac{1}{k} \sum_j (\Delta_j, \delta_j)
\]
(4.5)
And from \(\mathcal{C}\)’s convexity,
\[
\forall j (\Delta_j, \delta_j) \in \mathcal{C} \implies \frac{1}{k} \sum_j (\Delta_j, \delta_j) \in \mathcal{C}
\]
(4.6)
Denote \((\hat{\Delta}, \hat{\delta}) = \frac{1}{k} \sum_j (\Delta_j, \delta_j)\) and rewrite Eq. (4.5) and (4.6):
\[
(\hat{A}, \hat{c}) = (\hat{A}_0, \hat{c}_0) + (\hat{\Delta}, \hat{\delta})
\]
\[
\forall j (\Delta_j, \delta_j) \in \mathcal{C} \implies (\hat{\Delta}, \hat{\delta}) \in \mathcal{C}
\]
Figure 4.3: Sliding and infinite window models. When $A^j$ overlaps $A_0^j$, $\Delta^j = A^j - A_0^j = \sum_{\text{new}} x_i x_i^T - \sum_{\text{old}} x_i x_i^T$.

Substitute in Eq. (4.4) to finally obtain:

$$\forall j (\Delta^j, \delta^j) \in C \implies \| (\hat{A}_0 + \hat{\Delta})^{-1}(\hat{\epsilon}_0 + \hat{\delta}) - \hat{A}_0^{-1}\hat{\epsilon}_0 \| = \| \hat{\beta} - \hat{\beta}_0 \| = \| \beta - \beta_0 \| \leq \epsilon$$

which completes the proof.

4.2.3 Infinite and Sliding Window

We differentiate between two different variations for computing the global model: sliding window and infinite window. In the sliding window model, $\beta$ is computed from the last $W$ samples seen at each node, and similarly $\beta_0$ is computed from the last $W$ samples before sync. Conversely, in the infinite window model $\beta$ is computed over all observations seen thus far, while $\beta_0$ is computed from all observations seen until last sync. Figure 4.3 illustrates these two models. Though the sliding window is clearly more practical, the infinite window model may be useful in some settings and so we discuss both.

Sliding Window In the sliding window model each node computes $A^j$ from the $W$ samples seen at node $j$, while $A_0^j$ (and hence $\hat{A}_0$) is built from the last $W$ samples before sync. Computing $\Delta^j$ and $\delta^j$, however, requires substracting observations that left the sliding window. If $A_0^j, c_0^j$ and $A^j, c^j$ do not overlap (Figure 4.3, top), then clearly $\Delta^j = A^j - A_0^j$ and $\delta^j = c^j - c_0^j$. It is also possible, however, that the current window overlaps the window used to build $\beta_0$. Figure 4.3 (middle) illustrates this case: $\Delta^j, \delta^j$ become the sum of new samples from $A^j, c^j$ minus the sum of old (non-overlapping) samples from $A_0^j, c_0^j$.

The convex constraint $C$ on $(\Delta, \delta)$ for this model is:

$$\epsilon \| \hat{A}_0^{-1}\Delta \| + \| \hat{A}_0^{-1}\delta \| + \| \hat{A}_0^{-1}\Delta\beta_0 \| \leq \epsilon \quad , \quad (4.7)$$
Algorithm 4.1: Node $j$ update with new observation $x, y$.

1. $(A^j, c^j) \leftarrow (A^j + x^T x, c^j + x^T y)$
2. Insert new $x, y$ to head of sliding window.
3. Retrieve old $x, y$ exiting end of sliding window.
4. $(A^j, c^j) \leftarrow (A^j - x^T w_x, c^j - x^T w_y)$
5. $(\Delta^j, \delta^j) \leftarrow (A^j - A_0^j, c^j - c_0^j)$
6. if $\epsilon \|\hat{A}_0^{-1}\Delta^j\| + \|\hat{A}_0^{-1}\delta^j\| + \|\hat{A}_0^{-1}\Delta^j\beta_0\| \leq \epsilon$ then
   7. Report violation to coordinator.
8. Receive new $\beta_0, \hat{A}_0^{-1}$ from coordinator.
9. $(A^j_0, c^j_0) \leftarrow (A^j, c^j)$

Algorithm 4.2: Coordinator violation resolution algorithm.

1. Poll all nodes for $A^j, c^j$.
2. Compute updated $\hat{A}_0^{-1}, \beta_0$ from $A^j, c^j$ and distribute.

where $\|A\|$ is the $L_2$ operator norm of the matrix $A$. The derivation of the convex constraint $C$ is shown in Section 4.2.6.

Alg. 4.1 shows the resulting monitoring algorithm each node runs. Note monitoring does not require any matrix inversions. Each node applies the local constraint from Eq. (4.7) to its own data. When a violation occurs at any node, it is reported to a coordinator node. The coordinator (Alg. 4.2) polls all nodes for their local data, computes a updated global model $\beta_0$ and distributes it to all nodes, along with updated $\hat{A}_0^{-1}$ used in the constraint. Monitoring then resumes. This is the simplest violation resolution protocol. We briefly discuss more sophisticated schemes [KCR06, KSA+14, GKS14] in Section 5. Similarly, the coordinator can use any algorithm to compute $\hat{A}_0^{-1}, \beta_0$.

Infinite Window In this model the local drifts of each node $i$ are $\Delta^i = A^i - A_0^i$ and $\delta^i = c^i - c_0$ as before, but $A^i$ and $c^i$ are computed from all observations ever seen at the node. We can use the same convex constraint from Section 4.2.3, but in this case $\Delta^i$ grows indefinitely, and so the condition $\|\hat{A}_0^{-1}\Delta^i\| < 1$ is not easy to satisfy, and may cause frequent synchronizations. Instead, we start from Eq. (4.9) and develop a more lenient constraint for this model. The resulting algorithm will be similar to Alg. 4.1, but without lines 2–4 and with the updated constraint in line 6. The coordinator algorithm is the same.

The convex constraint for the infinite window case is

$$\|\hat{A}_0^{-1}\delta\| + \|\hat{A}_0^{-1}c_0\| \leq \epsilon \quad .$$  \hspace{1cm} (4.8)

Section 4.2.7 details its derivation.

Note that $\delta$ accumulates more samples as time passes, while $\hat{A}_0$ remains fixed. As $\delta$’s “weight” (number of samples) grows beyond $\hat{A}_0$’s, the constraint no longer holds.
and synchronization is needed. One way to avoid this is to replace $\|\hat{A}_0^{-1}\delta\|$ in Eq. 4.8 with $\|\Delta^{-1}\delta\|$, which is correct (using the same line of arguments as in Section 4.2.7). Alternatively, note that after each sync the samples from all $\delta^j$’s are added to the new $\hat{A}_0$, so its “weight” is roughly doubled. Thus $\hat{A}_0$’s weight grows exponentially, and synchronizations become increasingly rare.

4.2.4 Norm Constraint and the Sliding Window

The sliding window model constraint Eq. (4.7) requires $\|\hat{A}_0^{-1}\Delta j\| < 1$ (embodied as $\epsilon\|\hat{A}_0^{-1}\Delta j\| + \cdots \leq \epsilon$). This requirement depends only the independent variables $X^j$, and does not depend in any way on the dependent variable $y^j$. It is quite possible that $\beta$ is close to $\beta_0$, yet the norm constraint is violated, incurring extra communication. Fortunately, for many reasonable data distributions, if window size $W$ is linear in the number of independent variables $m$ then the norm constraint is satisfied almost surely. The details are in Section 4.2.8.

The analysis assumes that consecutive observations in the data stream are independent. Consider, however, the case where some variables come from an over-sampled sensor, or measure slowly changing phenomena. In such cases $\|\Delta\|$ grows faster, linear in the number of identical observations, and will overwhelm $\hat{A}_0^{-1}$ faster. This can result in more frequent violations of the constraint, hence more communication. Such cases can be mitigated by increasing the window size $W$, subsampling (since data changes slowly anyway), or by the use of generalized least squares [Hay00] with an appropriate scaling matrix for the time series process (Section 4.2.5).

4.2.5 Regularization and Variants

Our scheme generalizes very well to more sophisticated least squares variants [Hay00]. We show two examples.

In regularized least squares the minimized function includes a regularization term to mitigate the effects of outliers and avoid overfitting. A commonly used form is Tikhonov regularization, also known as ridge regression, which finds $\beta$ that minimizes $\|X\beta - y\|^2 + \|R^T R \beta\|^2$, where $R$ is a suitable regularization matrix $R$. For $R = 0$ the problem reduces to ordinary least squares, and for $R = \lambda I$ it reduces to $L_2$ regularization. The optimal solution to this problem is

$$\beta = (X^T X + R^T R)^{-1} X^T y.$$  

This solution is quite similar to Eq. (4.1) and indeed we can monitor it using the same technique: compute $B^j_0 = A^j_0 + \frac{1}{k} R^T R$ and the resulting $B_0, \hat{B}_0$, and use them in Lemma 4.2.1 instead of $A^j_0, A_0, \hat{A}_0$.

Similarly, generalized least squares handles correlated measurements and errors by minimizing the Mahalanobis distance $(Y - X\beta)^T S^{-1}(Y - X\beta)$, where $S$ is the covariance
matrix of the residuals (errors). Again, GLS reduces to OLS if $S = I$. As before, we can monitor the optimal solution

$$\beta = (X^T S^{-1} X)^{-1} X^T \tilde{y},$$

where $\tilde{y} \triangleq S^{-1} y$.

by monitoring $B = \sum_i S^{-1} x_i x_i^T$ and $d = \sum_i x_i \tilde{y}_i$. GLS is particularly useful in time series analysis, where $S$ is the process’ structured covariance (or autocorrelation) matrix [Sau99].

**4.2.6 Deriving the Sliding Window Constraint**

To find a convex subset $C$ satisfying the condition of Eq. (4.4), we first review some notions and well-known results on norms of real matrices [Rom95]. We use the $L_2$ norm throughout.

**Definition 4.2.2.** Let $A$ be a matrix. Its *operator norm*, or *spectral norm*, hereafter just *norm*, is defined as

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}.$$  

It follows that for a matrix $A$ and vector $x$, $\|Ax\| \leq \|A\| \|x\|$.

Moreover, for any two matrices $A, B$: $\|A + B\| \leq \|A\| + \|B\|$ and $\|AB\| \leq \|A\| \|B\|$. If $A$ is a symmetric matrix, $\|A\| = \max_i |\lambda_i|$ where $\lambda_i$ are the eigenvalues of $A$. Additionally, if $A, B$ are symmetric then $\|AB\| = \|BA\|$.

**Lemma 4.2.3.** For square $A$ with $\|A\| < 1$, $(I-A)$ is invertible, the inverse being the Neumann series [Mil81]: 

$$(I - A)^{-1} = I + A + A^2 + A^3 + \ldots.$$  

**Lemma 4.2.4.** If $A$ is square and $\|A\| < 1$, then

$$\|(I + A)^{-1}\| = \|I - A + A^2 - A^3 + \ldots\| \leq \frac{1}{1 - \|A\|}.$$  

**Proof.** Apply Lemma 4.2.3 and the triangle inequality:

$$\|(I + A)^{-1}\| = \|I - A + A^2 - A^3 + A^4 - \ldots\| \leq \|I\| + \|A\| + \|A^2\| + \cdots \leq \frac{1}{1 - \|A\|},$$

since it is the sum of a geometric series.

We begin by subtracting and adding $(\hat{A}_0 + \Delta)^{-1} \hat{c}_0$ to the bounded expression in Eq. (4.4):

$$\|(\hat{A}_0 + \Delta)^{-1}(\hat{c}_0 + \delta) - \hat{A}_0^{-1} \hat{c}_0\| =$$

$$\|(\hat{A}_0 + \Delta)^{-1}\delta + (\hat{A}_0 + \Delta)^{-1} - \hat{A}_0^{-1}\| \hat{c}_0\|.$$
Applying the triangle inequality, we obtain:
\[
\underbrace{\| (\hat{A}_0 + \Delta)^{-1}\delta \|}_{E_1} + \underbrace{\left\| \left( (\hat{A}_0 + \Delta)^{-1} - \hat{A}_0^{-1} \right) \hat{c}_0 \right\|}_{E_2}.
\] (4.9)

Next, note that
\[
(\hat{A}_0 + \Delta)^{-1} = \left( \hat{A}_0 \left( I + \hat{A}_0^{-1} \Delta \right) \right)^{-1} = \left( I + \hat{A}_0^{-1} \Delta \right)^{-1} \hat{A}_0^{-1}
\]
and, assuming \( \| \hat{A}_0^{-1} \Delta \| < 1 \), we apply Lemma 4.2.3 to obtain:
\[
(\hat{A}_0 + \Delta)^{-1} = \left( I - \hat{A}_0^{-1} \Delta + \hat{A}_0^{-1} \Delta \hat{A}_0^{-1} \Delta - \ldots \right) \hat{A}_0^{-1}
\] (4.10)
\[
= \hat{A}_0^{-1} - \hat{A}_0^{-1} \Delta \hat{A}_0^{-1} + \hat{A}_0^{-1} \Delta \hat{A}_0^{-1} \Delta \hat{A}_0^{-1} - \ldots
\] (4.11)

Note the assumption \( \| \hat{A}_0^{-1} \Delta \| < 1 \) is not trivial, and we discuss it in Section 4.2.4 and Appendix 4.2.8.

We now apply Eq. (4.10) and Lemma 4.2.4 to \( E_1 \) in Eq. (4.9):
\[
E_1 = \| (\hat{A}_0 + \Delta)^{-1}\delta \|
= \left\| \left( I - \hat{A}_0^{-1} \Delta + \hat{A}_0^{-1} \Delta \hat{A}_0^{-1} \Delta - \ldots \right) \hat{A}_0^{-1} \delta \right\|
\leq \| I - \hat{A}_0^{-1} \Delta + \hat{A}_0^{-1} \Delta \hat{A}_0^{-1} \Delta - \ldots \| \| \hat{A}_0^{-1} \delta \|
\leq \frac{\| \hat{A}_0^{-1} \delta \|}{1 - \| \hat{A}_0^{-1} \Delta \|}
\] (4.12)

Similarly, we apply Eq. (4.11) to \( E_2 \):
\[
E_2 = \left\| \left( (\hat{A}_0 + \Delta)^{-1} - \hat{A}_0^{-1} \right) \hat{c}_0 \right\|
= \left\| \left( \hat{A}_0^{-1} - \hat{A}_0^{-1} \Delta \hat{A}_0^{-1} + (\hat{A}_0^{-1} \Delta)^2 \hat{A}_0^{-1} - \cdots - \hat{A}_0^{-1} \right) \hat{c}_0 \right\|
= \left\| \left( I + \hat{A}_0^{-1} \Delta - (\hat{A}_0^{-1} \Delta)^2 + \ldots \right) \hat{A}_0^{-1} \Delta \beta_0 \right\|
\] (4.13)

Applying Lemma 4.2.4 to Eq. (4.13) we obtain:
\[
E_2 \leq \left\| I + \hat{A}_0^{-1} \Delta - (\hat{A}_0^{-1} \Delta)^2 + \ldots \right\| \| \hat{A}_0^{-1} \Delta \beta_0 \|
\leq \frac{\| \hat{A}_0^{-1} \Delta \beta_0 \|}{1 - \| \hat{A}_0^{-1} \Delta \|}
\] (4.14)

Substituting Eq. (4.12) and (4.14) in Eq. (4.9) and rearranging, we arrive at the
convex constraint $C$ on $(\Delta, \delta)$:

$$\epsilon \| \hat{A}_0^{-1}\Delta \| + \| \hat{A}_0^{-1}\delta \| + \| \hat{A}_0^{-1}\Delta \beta_0 \| \leq \epsilon \ .$$

(4.15)

This convex constraint allows us to apply Lemma 4.2.1. Satisfying Eq. (4.15) guarantees Eq. (4.4) is also satisfied, since the bounded expression is larger. Moreover, this bound is a subset of $\| \hat{A}_0^{-1}\Delta \| < 1$, a necessary condition for correctness, meaning we don’t have to check it explicitly.

4.2.7 Deriving the Infinite Window Constraint

A matrix $A$ is positive definite, denoted $A \succ 0$, if $x^TAx > 0$ for all non-zero vectors $x$. This implies a partial ordering of square matrices: we denote $A \succ B$ if $A - B \succ 0$. Note $A \succ B \succ 0 \implies \|A\| > \|B\|$. Moreover, $A \succ B \succ 0 \implies (A - B)^{-1} \succ 0$. Finally, observe that $\| (A + B)^{-1}u \| \leq \| A^{-1}u \|$, since $A + B \succ A$ and therefore $A^{-1} \succ (A + B)^{-1}$. Similarly, $\| (A + B)^{-1} - A^{-1} \| u \| \leq \| A^{-1} u \|$.

We apply the above to Eq. (4.9). Note that by construction, $\Delta j = \sum_{i \in S_j} x_i x_i^T$, where $S_j$ is the set samples seen by node $j$ since the last sync time, is symmetric and positive definite. Similarly, $\hat{A}_0$ is symmetric positive definite by construction. Thus, $E_1 = \| (\hat{A}_0 + \Delta)^{-1}\delta \| \leq \| \hat{A}_0^{-1}\delta \|$, and $E_2 = \| (\hat{A}_0 + \Delta)^{-1} - \hat{A}_0^{-1} \| c_0 \| \leq \| \hat{A}_0^{-1}c_0 \|$.

The final convex constraint for the infinite window case is therefore

$$\| \hat{A}_0^{-1}\delta \| + \| \hat{A}_0^{-1}c_0 \| \leq \epsilon \ .$$

(4.16)

4.2.8 Window Size And Dimensions

We will show that sliding window $W$ linear in $m$ will avoid overwhelming $\| \hat{A}_0^{-1}\Delta j \|$ in Eq. (4.7).

For any matrix $A$, denote its largest and smallest eigenvalues by $\lambda_{\text{max}}(A)$ and $\lambda_{\text{min}}(A)$. Recall that $\hat{A}_0 = \frac{1}{k} A_0$ and that in the sliding window model\textsuperscript{2}, $\Delta j = A^j - A_0$, and that all these matrices are symmetric by construction. Moreover, if $A$ is symmetric then $\|A\| = \sqrt{\lambda_{\text{max}}(A^T A)} = |\lambda_{\text{max}}(A)|$. Finally, $\lambda_{\text{max}}(A^{-1}) = \frac{1}{\lambda_{\text{min}}(A)}$, and therefore

$$\| \hat{A}_0^{-1} \| = \left\| \frac{1}{k} A_0^{-1} \right\| = \frac{k}{|\lambda_{\text{min}}(A_0)|} = \frac{k}{\lambda_{\text{min}}(A_0)} \ .$$

Applying the above to the norm constraint:

$$\| \hat{A}_0^{-1}\Delta j \| \leq \| \hat{A}_0^{-1}\|\|\Delta j\| = \frac{k}{\lambda_{\text{min}}(A_0)}|\lambda_{\text{max}}(A^j - A_0)|$$

$$\leq \frac{k}{\lambda_{\text{min}}(A_0)}|\lambda_{\text{max}}(A^j) - \lambda_{\text{min}}(A_0^j)| \ .$$

(4.17)

\textsuperscript{2}We discuss the worst case, when $A^j, A_0^j$ do not overlap. When they do, $\Delta j$’s effective window size is less than $W$.  

62
The last step is obtained from [KT01]:

\[ A, B \text{ symmetric} \implies \lambda_{\max}(A + B) \leq \lambda_{\max}(A) + \lambda_{\max}(B) \]

and since \( \lambda_{\max}(-B) = -\lambda_{\min}(B) \).

The bound in Eq. (4.17) depends on the distribution of the data. Assume the elements of \( X \) are drawn i.i.d from \( N(0,1) \), then the Marchenko-Pastur law [MP67] limits the spectrum of the Wishart matrix \( X^T X \).

**Lemma 4.2.5.** Let \( X \in \mathbb{R}^{W \times m} \) drawn as above such that \( \frac{m}{W} \) converges to 0 < \( b \leq 1 \) as \( W \) and \( m \) grow to infinity.\(^3\) Let \( M = \frac{1}{W} X^T X \), and denote its largest and smallest eigenvalues by \( \lambda_{\max}(M) \), \( \lambda_{\min}(M) \). Then almost surely

\[
\lambda_{\max}(M) \to (1 + \sqrt{b})^2, \quad \lambda_{\min}(M) \to (1 - \sqrt{b})^2.
\]

Bai and Yin [BY93] extended this result to any zero-mean distribution with unit variance and finite fourth moment [RV10]. These can be achieved using [GKS14], for example.

Note \( A_0 = \sum^k_{j=1} A^j_0 = \tilde{X}_0^T \tilde{X}_0 \), where \( \tilde{X}_0 \in \mathbb{R}^{kW \times m} \) is the concatenation of all local data matrices. Applying Lemma 4.2.5 to Eq. (4.17), we obtain

\[
\frac{k|\lambda_{\max}(A^j) - \lambda_{\min}(A^j_0)|}{\lambda_{\min}(A_0)} = \frac{kW|\lambda_{\max}(\frac{1}{W} A^j) - \lambda_{\min}(\frac{1}{W} A^j_0)|}{kW\lambda_{\min}(\frac{1}{W} A_0)} = \frac{|\lambda_{\max}(\frac{1}{W} A^j) - \lambda_{\min}(\frac{1}{W} A^j_0)|}{\lambda_{\min}(\frac{1}{W} A_0)},
\]

which converges almost surely to

\[
f_k(b) \triangleq \frac{[(1 + \sqrt{b})^2 - (1 - \sqrt{b})^2]}{(1 - \sqrt{\frac{b}{k}})^2} = \frac{4\sqrt{b}}{\left(1 - \sqrt{\frac{b}{k}}\right)^2}.
\]

In practice, Eq. (4.7) is the sum of 3 norms, so we require \( \| \hat{A}_0^{-1} \Delta^j \| < \frac{1}{3} \). Solving 0 < \( f_k(b) < \frac{1}{3} \) for \( b \) with \( k > 1 \) yields

\[
\frac{m}{W} \leq g_k \triangleq 72k^2 + k - 24k^2 - 4\sqrt{3} \sqrt{108k^4 + 15k^3 - 72k^2} - k^2.
\]

For given \( k > 1 \), selecting \( W \geq \frac{m}{g_k} \) guarantees \( \| \hat{A}_0^{-1} \Delta^j \| < \frac{1}{3} \) almost surely. The constant \( \frac{1}{g_k} \) grows slowly: for \( k = 2 \), the window size \( W \) must be at least \( \frac{1}{g_2} \approx 111.06m \); for \( k = 10 \), \( W \geq \frac{1}{g_{10}} \approx 129.02m \); and for \( k = 100 \), \( W \geq \frac{1}{g_{100}} \approx 139.22m \). In fact, \( g_k \) converges: \( \lim_{k \to \infty} g_k = \frac{1}{144} \), so a window size of \( W \geq 144m \) is sufficient for any \( k \).

\(^3\)Trivially, if \( W = \frac{m}{b} \).
4.3 Evaluation

We evaluated performance of our monitoring algorithm, DILSQ, for Distributed Least SQuare monitor, using simulations with two synthetic and two real-world distributed datasets. For each dataset, we run through the data, simulate the nodes (Alg 4.1) and the coordinator (Alg 4.2), count messages, and keep track of the resulting true models $\beta$ and the current monitored models $\beta_0$. Our simulations use discrete time (rounds), and we use the OLS variant of our algorithm with sliding window (Section 4.2.3), except for the gas sensor dataset which uses the GLS variant (Section 4.2.5).

Our baseline is the naive algorithm, where each node sends every new measurement to a centralized location each round. We compare DILSQ to the $T$-periodic algorithm, denoted PER($T$), a simple sampling algorithm that sends updates every $T$ rounds. Though PER cannot guarantee maximum error, it can achieve arbitrarily low communication.

Our main performance metric is communication, measured in normalized messages – the average messages sent per round by each node [BDG11]. Note that communication of the naive algorithm is always 1. When calculating and reporting results, we skip the first (incomplete) window (or the first epoch for the drift dataset described below).

DILSQ is designed to communicate as little as possible while always maintaining maximum model error below $\epsilon$. It guarantees maximum model error below the user-selected threshold $\epsilon$, but PER does not. Hence, when comparing the two, we find a posteriori the maximum period $T$ (hence minimum communication) for which the maximum error of PER($T$) is equal or below that of DILSQ. Note this gives PER an unrealistic advantage. First, in a realistic setting we cannot know a priori the optimal period $T$. Second, model changes in realistic settings are not necessarily stationary: the rate of model change may evolve, which DILSQ will handle gracefully while PER cannot.

4.3.1 Synthetic Datasets

We use two types of synthetic dataset. In the fixed dataset, the true model $\beta_{true} \in \mathbb{R}^m$ is fixed, with elements drawn i.i.d from $N[0, 1]^4$. We generate $R$ rounds with $k$ nodes, each receiving at each round a new data vector $x$ of size $m$ and scalar $y$. $x$ is drawn i.i.d from $N(0, 1)$, and $y = x^T \beta_{true} + n$ where $n \sim N(0, \sigma^2)$ is Gaussian white noise of strength $\sigma$. In the drift dataset the coefficients of $\beta_{true}$ change rapidly during 25% of one epoch, and are fixed during the rest of the epoch. We generate observations for $E$ epochs using the same procedure. For each experiment we generate new data.

Default parameter values are $k = 10$ nodes, $m = 10$ dimensions, noise magnitude $\sigma = 10$ (to generate interesting results given the large window), window size $W = 1300$

\[4\] Therefore $\|\beta\|^2 \sim \chi^2_m$
Figure 4.4: DILSQ model error (black) and syncs (bottom vertical lines) per round, compared to PER(100) error (green), for \( k = 10 \) simulated nodes with \( m = 10 \) dimensions, and threshold \( \epsilon = 1.35 \). Both algorithms reduce communication to 1%, but DILSQ only syncs when \( \beta \) changes (bottom purple line shows \( ||\beta|| \)). PER(100) syncs every 100 rounds, but is unable to maintain error below the threshold (dashed horizontal line).

and maximum error threshold \( \epsilon = 0.5 \), which is quite strict\(^5\). We generate \( R = 16900 \) rounds for the fixed dataset, or \( E = 5 \) epochs of 3900 rounds each for drift dataset.

Figure 4.4 shows the behavior of the monitoring algorithm over such a simulation on the drift dataset with \( \epsilon = 1.35 \) and 3 epochs. For this configuration, DILSQ achieves communication of 0.01 messages per node per round, and the model error is always below the threshold. Conversely, the equivalent PER(100) algorithm is unable to maintain the error below the threshold, which would require a higher update frequency. When model changes in \( \beta \) are large and frequent DILSQ performs more synchronizations, resulting in updated \( \beta_0 = \beta \) that decreases the error. When \( \beta \) is stable (it is never truly constant due to noise), synchronizations are much rarer. The periodic algorithm, on the other hand, synchronizes every 100 iterations even during the periods where \( \beta \) changes very little.

Effect of Threshold

Figure 4.5 shows the communication required for different threshold levels for the DILSQ algorithm, and the minimal communication required to match DILSQ using the PER algorithm with optimal period, as discussed above. For the fixed model dataset (Figure 4.5(a)) neither algorithm needs to sync very often to provide an accurate estimate. Had there been no noise, a single initial synchronization would have been sufficient, regardless of threshold. Note that for more permissive threshold values (or smaller noise magnitude \( \sigma \)) DILSQ achieved zero communication (beyond initial sync) for the fixed dataset (not shown in this log-scale figure).

Performance on the drift dataset (Figure 4.5(b)) is more interesting. When \( \epsilon \) is

\(^5\)Given that elements of both \( \beta_0 \) and \( \beta \) are i.i.d \( N(0, \sigma) \), then \( \frac{\|\beta - \beta_0\|}{\sqrt{2m}} \sim \chi_m \). The probability that a random \( \epsilon = \beta - \beta_0 \) will overwhelm \( \epsilon \) is \( P = 1 - \text{CDF}_{\chi_m}(\frac{\epsilon}{\sqrt{2m}}) > 1 - 10^{-8} \).
very strict, both algorithms perform roughly the same, with normalized messages of 0.25–0.75. As $\epsilon$ grows DILSQ develops an increasing advantage over PER with optimal period. The optimal period must be low enough to match the quickly changing model, and is wasteful on the intervals where $\beta$ is quiescent. For our dataset, $\beta_{true}$ is constant during roughly 75% of each epoch. For datasets with larger quiescent periods (or smaller window), the advantage of DILSQ will be even larger.

Scalability

Figure 4.6 explores how performance of DILSQ scales with different parameters. Figure 4.6(a) shows communication for different values of the number of nodes $k$. We observe communication increases slowly, remaining below 10% even with 500 nodes. Figure 4.6(b) shows normalized messages obtained at different noise magnitudes. Below a certain level of noise, communication is fairly constant, reflecting the choice of threshold $\epsilon$. At lower values of noise (not shown), DILSQ requires no communication for the fixed model dataset, beyond the first window of $W$ observations. Figure 4.6(c) compares communication with the number of independent variables $m$ on the drift dataset, confirming our analysis in Section 4.2.4. When window size $W$ is fixed, communication grows linearly with dimension $m$. However, if $W$ grows linearly with $m$, we see that communication remains very low (and in fact decreases a little). In both cases we keep epoch length to be $3W$ to maintain the same rate of change of $\beta$ across the window.

Similarly, Figure 4.6(d) shows what happens when the window size is too small compared to the value predicted in Section 4.2.4. It depicts communication obtained on the fixed dataset, as a function of window size $W$. As window size decreases below 144$m$ (see Appendix 4.2.8), constraint violations are more frequent as data periodically overwhelms the norms in Eq. (4.7). As we will see below, in practical settings a much
Figure 4.6: Communication vs. different parameters for the fixed (green) and drift (black) datasets. (a) shows DILSQ is scalable: communication increases slowly with number of nodes. (b) shows communication is fairly constant when noise is small ($\sigma < 10$). Comm. is zero for fixed model at low noise (not shown). (c) shows the required window size $W$ is linear in $m$: communication does not increase when $W$ is suitably sized (purple). (d) shows performance with fixed dataset. If $W < 144m$ data periodically saturates the norms in Eq. (4.7).
lower $W$ can be used, since data values have finite ranges, change slowly, and model changes are more frequent.

### 4.3.2 Traffic Monitoring

Consider the following interpolation problem: given periodical traffic measurements (average velocity every minute) from a small number of sensors embedded along a long road, we wish to infer the current average velocity at every point along the road. We aim to solve this problem using polynomial regression. Note that in this case we have no good way to measure the true error of our model, since we do not have sensors in other locations. Moreover, as Figure 4.1 (derived from the same data below) shows, monitoring model fit ($R^2$) is also problematic (Section 4.1.1). Instead, we rely on the fact that we can limit the model error $\|\beta - \beta_0\|$.

We used two weeks’ worth of velocity data collected during November 2014 from $k = 6$ sensors located along the Grenoble south ring in France [MLOCDWB14]. Reported measurements of each sensor are aggregated once per minute, and when measurements are unavailable average velocity was assumed to be unchanged. The road is composed of several sections, and we model it as the interval $[1,18]$, where the sensors are located at $l \in \{1,3,7,11,14,18\}$. The data from every sensor at location $l$ is always $x = [1,l,l^2,l^3,l^4]$, and $y$ is the velocity measured by the sensor. Given model $\beta$ built from measurements from the last hour ($W = 60$), the interpolated velocity at location $i \in [1,18]$ is $[1,i,i^2,i^3,i^4]\beta$.

Figure 4.7 shows the result of one such a prediction for 9am on Nov 1 2014, produced using $\epsilon = 25$. The pink dots represent average velocity measurements of each sensor between 8am to 9am. The dashed purple line is velocity interpolated using the exact least squares model $\beta$, while the black line is interpolated using DILSQ approximation $\beta_0$. Observe the resulting interpolation is fairly accurate, with errors below 10km/h across of range of interpolated positions.
Figure 4.8: Communication for DILSQ (black) and periodic algorithm (green) on the traffic dataset at different $\epsilon$ values.

Figure 4.9: Communication for DILSQ (black) and periodic algorithm (green) with same max model error on gas sensors data at different $\epsilon$ values.

Figure 4.8 explores the communication of DILSQ and matching PER with various levels of $\epsilon$, for window sizes 60 and 30. DILSQ is superior to PER across all ranges except the unrealistically strict $\epsilon = 5$ (average $\|\beta\|$ is roughly 100). For one hour window, DILSQ obtains 0.12 normalized messages for $\epsilon = 25$ used in Figure 4.7, and can reduce communication to 0.03 for $\epsilon = 85$. For a much smaller window size of half an hour, DILSQ requires more communication but still achieves considerable communication reduction: it requires 20% communication for $\epsilon = 25$ and can use as little as 5% for $\epsilon = 85$. Finally, we observe that the communication gap between DILSQ and PER increases considerably with smaller window size, as $\beta$ changes more quickly and is more sensitive to noise.

4.3.3 GLS on Gas Sensor Time Series

Data in this experiment consists of measurements collected by an array of 16 chemical sensors recorded at a sampling rate of 25Hz for 5 minutes, resulting in 7500 data points for each sensor. This dataset is described in [ZFF+15], and is publicly available [Lic13].
The original goal in [ZFF+15] is to identify certain gas classes given high-level frequency features. Since the original target variable is nominal and fixed throughout the run in each experiment, we defined a different regression problem. We divided the 16 sensors to \( k = 4 \) “nodes”, where in each node three sensors serve as the data \( x \) while the remaining sensor serves as the response \( y \). We also added a constant variable 1 to \( x \), to allow intercept in the model, hence \( m = 4 \). The regression task is therefore to predict the value of the 4th sensor in each node using the first three.

Note that in this setting measurement errors cannot be assumed to be independent, so an OLS models is ill-suited here. Instead, we assume errors are an AR(1) process and monitor the generalized least squares model [Hay00]. We used an AR(1) parameter value \( \phi = 0.95 \) for the autocorrelation matrix [Sau99]. Average \( \| \beta \| \) is 0.3, so we use \( \epsilon = 0.1 \), resulting in 0.17 normalized messages for DILSQ. We note that using an OLS model with the same \( \epsilon \) resulted in 1.15 normalized messages – the OLS model had to be updated very frequently as it was unstable.

Figure 4.9 shows the obtained communication for various \( \epsilon \) values in the range \([0.01,1]\). For \( \epsilon < 0.1 \) DILSQ is clearly superior: PER must communicate every round \((T=1)\) in order to match DILSQ, which achieves communication between 0.2 and 1 (for \( \epsilon = 0.01 \)). When \( \epsilon \) is more permissive, however, PER is superior and can obtain the same maximum error with less communication: with an extremely permissive \( \epsilon = 1 \), DILSQ requires 0.04 normalized messages while PER requires 0.015 for the same maximum error (though, of course, optimal \( T \) must be known \textit{a priori} to achieve this performance).
Chapter 5

Conclusions

In this work we developed novel communication-efficient algorithms for continuous estimation of three important functions in the setting of distributed data streams. Empirical evaluation on real datasets shows that communication can be reduced by up to three orders of magnitude, depending on the estimated function and the data.

Variance and Outlier Detection  We first described a communication-efficient distributed variance estimation scheme with guaranteed approximation bounds. We then used it and other streaming techniques to adapt the latent fault detector in [GSBB12] to a distributed setting: safe zones are used to monitor the data and scale it to approximately uniform variance, and sketching reduces the amount of data that must be sent. Nodes utilize already available information to avoid communication by predicting future data behavior. The coordinator distributes available slack: some nodes are allowed more deviation, depending on local data behavior at every node. While designed for scale-out, load-balanced systems, the resulting outlier detector can be applied in many cases where distributed multivariate time series are expected to have common normal behavior. We also described a communication-efficient distributed variance estimation scheme with guaranteed approximation bounds. This, in particular, may have many applications beyond outlier detection. Additionally, the online adaptation incrementally computes current outliers. The reduced dimensionality of the sketch means that less communication, memory and computation are required to update the p-values.

Our experiments on data from a real-world system show that adapted detector reduces communication by an order of magnitude (11–13% bandwidth reduction with below 1% new false positives or negatives). It is also resilient to scaling errors and sketching. In practice, performance seldom drops even when the variance approximation bounds are large. Similarly, even at small sketch sizes (216 counters reduced to 5 dimensions), the adapted latent fault detector performs close to the centralized version.

There are several avenues for improvement. Even with slack and prediction, 92.5% of violations remain local violations; there is room to explore different slack schemes, with
more complex policies. Additionally, nodes could resolve violation locally by exchanging slack directly, avoiding synchronization [KSA+14]. Finally, we can exploit the robustness to scaling errors by avoiding frequent synchronizations.

**Entropy** We then presented CIDER, a practical, communication-efficient entropy estimator for distributed streams. Unlike previous work, CIDER provides deterministic approximation bounds, and supports both insertions and deletions, allowing for variable-sized or time-based windows. Like previous work, it allows the use of entropy sketches to both reduce message size and to handle very high cardinality (\(k\) in the billions). We also extend geometric monitoring with variable-sized windows using a novel slack (drift) re-weighting scheme, which can be used in other geometric monitoring algorithms.

CIDER adapts to the underlying data and the user-specified bounds, and provides no guarantees for reduced communication, other than the trivial. This is consistent with Arackaparambil et al. [ABC09] conclusion that nontrivial savings cannot be guaranteed when monitoring non-monotone functions with deletions. Nevertheless, in practice we see that CIDER achieves considerable communication savings on real datasets: up to three orders of magnitude with relaxed accuracy, and often two orders of magnitude with strict approximation bounds.

**Least-squares Regression** Finally, we presented DILSQ: the first communication-efficient monitoring algorithm for least-squares regression models that limits the error in model coefficients. By monitoring the deviation of the existing model from the true model, our approach is able to avoid costly communication and model recomputations, while guaranteeing bounded model error. Each round, each node checks a simple local constraint on its own local data; if it is satisfied, communication is avoided. If not, violation is resolved by collecting data from all nodes and computing a new global model. Evaluation on real-world datasets shows a communication reduction of up to two orders of magnitude. Simulations on synthetic datasets show our algorithm scales well with the number of nodes.

We emphasize that correctness of the local constraint is independent of network topology and the algorithm used to compute the model \(\beta_0\). Hence it is straightforward to adapt our method to other settings. First, the role of the coordinator can easily be replaced with convergecasting [BK08, WBK09], yielding a peer-to-peer monitor. Alternatively, our distributed monitoring approach can easily be combined with an efficient distributed computation technique, enjoying the best of both worlds: the current model can be computed during sync using any of several existing algorithms, be they exact, iterative, or distributed [GBT+04, LS06]. Similarly, our method is compatible with recent communication reduction techniques from the field of distributed streams, such as reference point prediction [GDG+12], individualized constraints or slack [KCR06, GKS14], and local violation resolution [KSA+14]. We leave such extensions for future work.
Bibliography


[AQI16] Technical assistance document for the reporting of daily air quality – the air quality index (AQI), May 2016.


73


S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using


The chapter starts with understanding the importance of data sets and their role in the communication process. It discusses the problem of data transmission and the need for efficient algorithms to compute the entropy of the data. The chapter introduces a new algorithm that is effective in computing the entropy of the data, even in the presence of noise. The algorithm is non-deterministic, meaning it cannot predict the entropy value but provides a guaranteed estimate that is close to the actual value. The algorithm is also efficient in terms of communication, as it requires only a small amount of communication to compute the entropy.

This algorithm is particularly useful when dealing with dynamic data sets, where the data can change over time. The algorithm is also efficient in terms of communication, as it requires only a small amount of communication to compute the entropy.

The chapter then discusses the use of linear regression models to predict the entropy of the data. Linear regression models are widely used in many fields, such as economics, finance, and engineering. The chapter introduces an adaptive algorithm that is able to handle multidimensional data and is effective in predicting the entropy of the data.

The chapter concludes by discussing the importance of understanding the behavior of the data and the need for efficient algorithms to compute the entropy of the data. The chapter provides an overview of the different methods used to compute the entropy of the data and discusses the advantages and disadvantages of each method.

The chapter concludes by discussing the importance of understanding the behavior of the data and the need for efficient algorithms to compute the entropy of the data. The chapter provides an overview of the different methods used to compute the entropy of the data and discusses the advantages and disadvantages of each method.
Technion - Computer Science Department - Ph.D. Thesis PHD-2017-11 - 2017

Technique generally useful in communication is geometric monitoring (threshold crossing) used by network engineers. Analysis of monitoring is often possible to do (domain) through a threshold crossing, a half-width (target), or another metric within communication functions, as well as methods for monitoring the presence of anomalies and other factors.

On the other hand, finding the best solution for communication is always a challenge. This is especially true in modern networks, where the number of users and devices is constantly increasing. The traditional methods of detection are often not enough to detect these anomalies in a timely manner.

To address these challenges, researchers have developed new algorithms and techniques for monitoring and detecting anomalies. These techniques are often based on statistical methods, such as variance and entropy, which are used to detect anomalies in real-time communication systems.

The goal of these techniques is to detect anomalies in real-time communication systems, while minimizing the impact on performance. This is achieved through the use of efficient algorithms and techniques, such as least-squares regression and dynamic load balancing, which are designed to detect anomalies in real-time communication systems.

In conclusion, the detection of anomalies in real-time communication systems is a challenging task, but with the development of new algorithms and techniques, it is possible to detect anomalies in a timely and efficient manner.

References:

- Entropy
- Variance
- Least-squares regression
- Dynamic load balancing
- Latent fault detection
- Scale-out
- False positive rate
תקציר

תוארגים קלאסיים לכריתת מידע לרוב מניחים שהקל משולב בתחום זול הצמוד, ומ IPC ערכים отдלייה ערכים ממומרים למטריעות (throughput), שמעון לשעון וממשקים קרובים (mini-batches) (machine learning) למודול, מתאימים וvronי ולוחצים ()はある כך באים בתוכן מספר מחשבים. במקביל, Büro, רוב הבודאיות לתוך מיני-אצטוס (machine learning) לומדים, ומניחים שה paddingHorizontal של המ_meter היחידה נ[OF] (on-line) לברית ואצל העלת האשראות של סילוק המתוכננים במפקדה.

[--

Technion - Computer Science Department - Ph.D. Thesis PHD-2017-11 - 2017--]


אלגוריתמים יעילים בתפקידת החלפת הנתונים בזירה שמוביליםcrm

תרבותonald ond משגרה

strstr של הפקולטה למדעי המחשב

דוקטור לפילוסופיה

משה גברל

המונח לתאר

הון שלן הוסביון – מוכן לתואר לפילוסופיה

תנ微型 התשע”ז

יולי 2017
אלגוריתמים יעילים בתקשות תכנית
נתונים בורמי مدير מבוזרים

משה גבל