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

Moshe Gabel, Assaf Schuster
Department of Computer Science
Technion – Israel Institute of Technology
Haifa, Israel
{*mgabel,assaf*}*@cs.technion.ac.il*

Daniel Keren
Computer Science Department
University of Haifa
Haifa, Israel
dkeren@cs.haifa.ac.il

***Abstract*—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 this work 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 describe a novel communication-efficient online distributed variance monitoring algorithm that provides a continuous estimate of the global variance within guaranteed approximation bounds. Using the 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.

***Keywords*-distributed computing; distributed processing; data analysis; time series analysis; fault detection;**

I. Introduction

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 [1], [2] 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 [3], and most require centralizing the data in some form. Rule-based failure detectors define a set of watchdogs [2] that trigger an alert whenever specified counters cross predefined thresholds. More advanced, supervised methods learn system behavior models from historical logs [1], [4], but are sensitive to workload changes and system updates [5]. Textual console log analysis in high performance computing [6], [7] 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 [8], [9] rely on domain insights and system knowledge, and therefore have limited applicability.

Recent approaches to the monitoring problem [3], [9], [10] 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 [3] 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 [4]. 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 [11], [12] and safe zones [13]–[15] to reduce communication and processing requirements by an order of magnitude, while preserving the framework's advantages.

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%.

II. Latent Fault Detection

In previous work [3] 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.

A. 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.

[1]We use a sampling period of 5 minutes, a good compromise between responsiveness and transmission delay.

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 $0 < \alpha < 1$.

B. 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 $0 < \alpha < 1$, 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 α, the null hypothesis is rejected, and the machine is flagged as suspicious.

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

C. The Sign Test

The sign test extends the classic statistical sign test to allow the simultaneous multivariate comparison of multiple

[2]Common in many systems and datacenters [6], [8], [9].

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

Algorithm 1: The sign test.

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

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

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 α. Algorithm 1 shows the sign test integrated into the framework. Derivation of $p(m)$ is described in [3].

III. Distributed Online Variance Monitoring

The framework in Section II-B 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 [13]–[16] 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.

[3]We select counters in advance, using the method described in [3].

```
Intialization: Run syncronization once.

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

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

Algorithm 2: Online Variance Monitoring.

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.

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.

A. 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 $\mu_i = \mathrm{E}[X_i(t)]$ is the mean of the last T values of X at node i, while the global mean $\mu = \mathrm{E}[X(t)]$ is the mean of the last T values across all nodes. Similarly, denote $\lambda_i = \mathrm{E}\left[X_i(t)^2\right]$, the local mean of the squares, and $\lambda = \mathrm{E}\left[X(t)^2\right]$ 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.

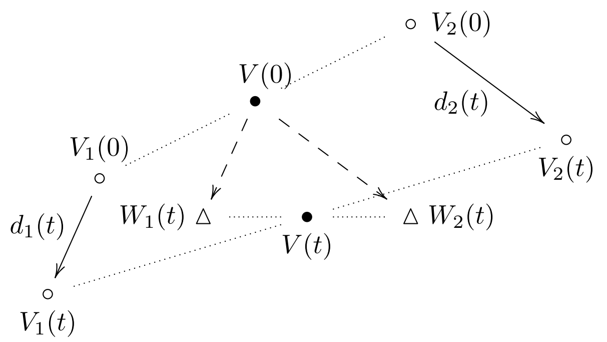
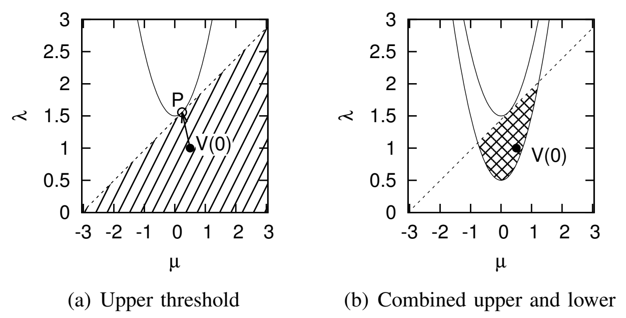


Figure 1. Global, local, reference, and drift vectors with two nodes. Note that $V(t) = \frac{V_1(t)+V_2(t)}{2} = \frac{W_1(t)+W_2(t)}{2}$.



(a) Upper threshold (b) Combined upper and lower

Figure 2. Safe zones for $L = 0.5, H = 1.5$ where $V(0) = (0.5, 1)$.

B. Defining Safe Zones

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

$$\mathrm{Var}(X) = \mathrm{E}\left[X^2\right] - \left(\mathrm{E}\left[X\right]\right)^2 = \lambda - \mu^2 \quad .$$

We therefore monitor the constraints $L \le \lambda - \mu^2 \le H$, 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 [13], 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 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:

$$V(t) = \frac{1}{M}\sum_i V_i(t) = V(0) + \frac{1}{M}\sum_i d_i(t) \quad . \qquad (2)$$

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) we arrive at the local constraints: if $\forall i, W_i \in G$ then $V(t) \in G$. Figure 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 $\mathrm{Var}(X) \ge L$. This corresponds to monitoring that $\lambda - \mu^2 \ge L$, 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 $b - a^2 \ge L$.

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 $\mathrm{Var}(X) \le H$. 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 $\mathrm{Var}(X) \le H$.

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 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(a) shows $V(0)$, P and the resulting safe zone, and Figure 2(b) shows the intersection with the safe zone for the lower limit $L = 0.5$.

C. 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 σ_0^2 be the last known global variance, given the current reference point $V(0)$: $\sigma_0^2 = \mathrm{Var}(X(0)) = \lambda(0) - \mu(0)^2$, and denote the current global variance $\sigma^2 = \mathrm{Var}(X) = \lambda - \mu^2$. We normalize counter data using the last known standard deviation σ_0 rather than σ. 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 $\frac{\sigma_0}{f} < \sigma < f\sigma_0$.

D. Safe Zone Violations

If one of the local constraints $W_j \in G$ is violated, it may be because $\mathrm{Var}(X)$ 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 μ and λ) 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 suboptimal. 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

E. Multiplicative Slack Distribution

The safe zones described in Section III-B 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 [17], [18], 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 β_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 β_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 $\mathcal{K}$ be the set of nodes that reported a local violation to the coordinator this round. If $|\mathcal{K}| \geq \frac{M}{4}$, 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 $\mathcal{L}$ 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 $w > 1.0$: $\forall i \in \mathcal{L} : \beta_i' \leftarrow \frac{\beta_i}{w}$. This extra slack is distributed to violating nodes: $\forall i \in \mathcal{K} : \beta_i' \leftarrow \beta_i + \frac{\gamma}{|\mathcal{K}|}$, where $\gamma = \left(1 - \frac{1}{w}\right)\sum_{i\in\mathcal{L}}\beta_i$ is the slack gained from balancing nodes. Finally, the coordinator sends the new slacks β_i' to the nodes in $\mathcal{K}$ and $\mathcal{L}$.

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.

F. Reference Point Prediction

Section III 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 [19], [20]. 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 Δt 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:

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

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 Δt, $V(0)$ 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, our safe zones guarantee that $\frac{\sigma'}{f} < \sigma < f\sigma'$, where σ' is the standard deviation from the predicted reference point $V'(t)$.

IV. Distributed Outlier Detection

Given the normalized counter, we now describe an online, communication-efficient version of the latent fault detector summarized in Section II. 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* [11], [12], 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 [21] or frequency moments [22]. 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 3 and explained in detail below.

```
Initialization:
Start variance monitoring on each counter.

Node $i$ at time point $t$:
foreach counter $c$ do
    Sample $X_c(t)$, the value of counter $c$ at time $t$.
    Update variance monitor for counter $c$ with $X_c(t)$.
    Scale $X_c(t)$ using estimated global mean and variance.
Let $x$ be the vector of normalized counter values.
Compute sketch of $x$ and send to coordinator.

Coordinator at time point $t$:
Handle violations in all variance monitors.
Receive sketches from all nodes.
Compute test function $S$ on received sketches.
foreach machine $m$ do
    Add most recent test function results to $v_m$.
    Subtract least recent test function results from $v_m$.
    Update $v_m$ sliding window.
    Calculate p-value $p(m)$ and flag if $p(m) \leq \alpha$
```

Algorithm 3: Adapted Latent Fault Detector.

A. 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 [21], [23]. The Johnson-Lindenstrauss Lemma [21], [23], [24] 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.

B. Sign Test on Linear Sketches

We first offer an intuitive, geometric explanation. The sign test function (1) from Section II-C 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 II-B, 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 $\|\frac{1}{T}\sum_t S\left(m, x(t)\right)\|$ 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: $\|\frac{1}{T}\sum_t S\left(m, Rx(t)\right)\|$, 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 1: Given $\epsilon \in (0, \frac{1}{2})$ and $\delta \in (0,1)$, let $k = O(^{\log\delta}/_{\epsilon^2})$ a large integer and $R \in \mathbb{R}^{k\times C}$ a suitable random projection (as in [21], [23]). Then with probability $1-\delta$, for any two unit vectors $u, v \in \mathbb{R}^C$,

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

Proof: From a variant of the Johnson-Lindenstrauss lemma [24, Theorem 3.1], with probability $1-\delta$, for any single arbitrary vector $x \in \mathbb{R}^C$,

$$(1-\epsilon)\|x\|^2 \leq \|Rx\|^2 \leq (1+\epsilon)\|x\|^2 \quad . \tag{3}$$

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 (3) we know $\|Ru\|^2 \leq (1+\epsilon)\|u\|^2 = (1+\epsilon)$ and $\|Rv\|^2 \leq (1+\epsilon)$. Thus,

$$\|Ru - Rv\|^2 \leq 2(1+\epsilon) - 2\langle Ru, Rv\rangle \quad . \tag{4}$$

Applying (3) to $x = u - v$:

$$\|Ru - Rv\|^2 \geq (1-\epsilon)\|u-v\|^2 = (1-\epsilon)\left(2 - 2\langle u,v\rangle\right) \quad . \tag{5}$$

Combining (4) and (5) and rearranging, we get

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

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

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

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 ($1-\delta$, where $\delta = O\left(^1/_{M^2}\right)$).

Note that the distortion introduced by the sketch (ϵ) 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 [3], 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 II-C is $2, \frac{2}{M-1}$*-bounded*[4]: 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 [3], yielding the same p-value in Algorithm 1. ■

C. Online Integration Using a Sliding Window

The integration phase in stage 2 of the framework in Section II-B computes $v_m = \frac{1}{T}\sum_t S\left(m, x(t)\right)$. Computing $S\left(m, x(t)\right)$ 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 $\frac{1}{T}S\left(m, \hat{x}(t)\right)$, and subtracting the least recent stored test result, $\frac{1}{T}S\left(m, \hat{x}(t-T-1)\right)$. 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 [3] $M = 4500$, $T = 288$, and with $k = 10$, the update rule is easy to compute on a single machine within the sampling period of 5 minutes.

V. Evaluation

Since the efficacy of the latent fault detector has already been established in previous work [3], 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 [3] (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. During simulation we run through the counter

[4]Definition from [3]

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 [3]. The window length was set to $T = 144$. Unless otherwise noted, our simulation includes the slack and reference point prediction mechanisms described in Section III. 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: $\frac{1}{M}\sum_{m\in\mathcal{M}}\log_{10}p(m) - \log_{10}p'(m)$. 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 $\alpha = 0.01$.

A. 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 3.

Figure 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 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 (figures omitted for lack of space). 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 V-B.

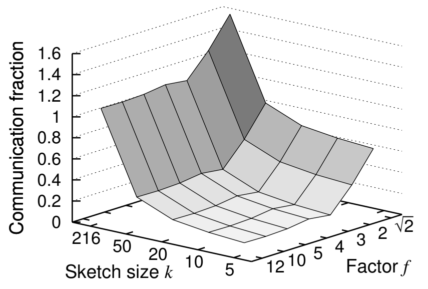
Figure 3(c) depicts the practical effect of different k, f combinations on the latent fault detector. Recall that detection error is defined as the percentage of machine classifications (normal, outlier) that differ between the centralized approach and the adapted, communication-efficient online approach. We've seen that k dominates p-value errors; this is confirmed by Figure 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.

B. Slack and Reference Point Prediction

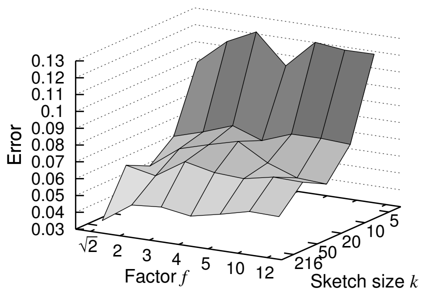
Section III describes slack and reference point prediction, two techniques to reduce communication, though with a possible increase in error. Figure 4 explores how they affect performance, using different variations of the algorithm.

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.

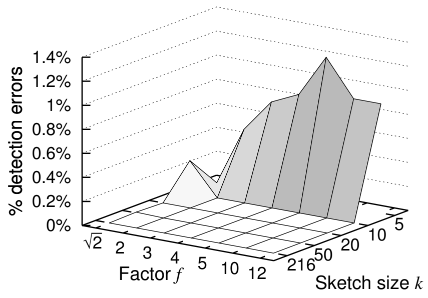
Figure 4(a) compares the four variants across different f values, with their median marked in a red line to represent typical behavior across the range of values. We focus on the impact of variance monitoring, so sketching is turned off ($k = C = 216$) and the relevant communication is not counted. Slack 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



(a) Communication fraction

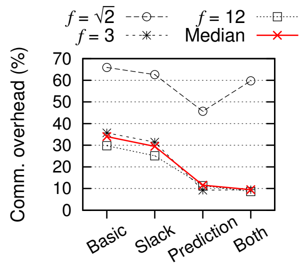


(b) Error

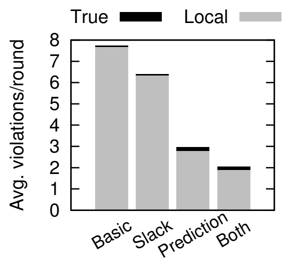


(c) Detection errors

Figure 3. Sign test performance at different k, f combinations, compared to centralized test.



(a) Communication overhead



(b) Avg violations with $f = 10$

Figure 4. Comparison of different variants of the monitoring algorithm.

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 (figure omitted for space). 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 4(b) 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.

VI. Related Work

Distributed Stream Monitoring: Early work on distributed stream monitoring focused on tailoring protocols for basic primitives by exploiting their specific properties. Babcock and Olston [18] monitor top-k items by tracking the differences from the current top-k and distributing slack. Bar-Or et al. [25] 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. [26] take advantage of the additive property of counts to alert if the global count crosses a specified threshold.

More recently, generic approaches approaches to monitoring distributed streams have received much attention. A recent survey by Cormode [17] formalizes the model and presents several results in this setting. The safe zone (SZ) approach we apply [13], [15] is a generalization of the geometric monitoring (GM) approach [16]. As with the SZ approach, GM defines local constraints that nodes can check rather to avoid communication. Sketching [11], [12] is a complementary general approach to stream processing. It focuses on reducing data size (e.g. using hash functions or random projections) without losing the ability to answer queries (e.g. number of distinct elements or inner product).

Distributed Outlier Detection: Our work 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. [27] 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. [28] 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 a maintain single model. TACO [29] 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. [30] 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. [31] 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.

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. [32]. 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 [33] and compute the variance over the sample. The most similar work is by Sharfman et al. [34], 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. Certain parts of this work have been previously published in a workshop [35].

VII. Conclusions

This work uses streaming techniques to adapt the latent fault detector in [3] 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 [14]. Finally, we can exploit the robustness to scaling errors by avoiding frequent synchronizations.

Acknowledgment

The research leading to these results has received funding from the European Union's Seventh Framework Programme under grant agreement № 255951. The authors thank Nir

Ailon for his valuable insights. Finally, we thank the IPDPS anonymous reviewers for their in-depth feedback.

References

[1] P. Bodík, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: Automated classification of performance crises," in *Proc. EuroSys*, 2010.

[2] M. Isard, "Autopilot: automatic data center management," *SIGOPS Oper. Syst. Rev.*, 2007.

[3] M. Gabel, A. Schuster, R.-G. Bachrach, and N. Bjorner, "Latent fault detection in large scale services," in *Proc. DSN*, 2012.

[4] G. Bronevetsky, I. Laguna, B. De Supinski, and S. Bagchi, "Automatic fault characterization via abnormality-enhanced classification," in *Proc. DSN*, 2012.

[5] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox, "Ensembles of models for automated diagnosis of system performance problems," in *Proc. DSN*, 2005.

[6] A. J. Oliner, A. Aiken, and J. Stearley, "Alert detection in system logs," in *Proc. ICDM*, 2008.

[7] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. SOSP*, 2009.

[8] M. P. Kasick, J. Tan, R. Gandhi, and P. Narasimhan, "Black-box problem diagnosis in parallel file systems," in *Proc. FAST*, 2010.

[9] S. Kavulya, R. Gandhi, and P. Narasimhan, "Gumshoe: Diagnosing performance problems in replicated file-systems," in *Proc. SRDS*, 2008.

[10] S. Kavulya, S. Daniels, K. Joshi, M. Hiltunen, R. Gandhi, and P. Narasimhan, "Draco: Statistical diagnosis of chronic problems in large distributed systems," in *Proc. DSN*, 2012.

[11] G. Cormode and M. Garofalakis, "Sketching probabilistic data streams," in *SIGMOD*, 2007.

[12] S. Muthukrishnan, "Data streams: Algorithms and applications," *Foundations and Trends in Theoretical Computer Science*, 2005.

[13] D. Keren, I. Sharfman, A. Schuster, and A. Livne, "Shape sensitive geometric monitoring," *Trans. Knowl. Data Eng.*, 2012.

[14] D. Keren, G. Sagy, A. Abboud, D. Ben-David, A. Schuster, I. Sharfman, and A. Deligiannakis, "Geometric monitoring of heterogeneous streams," *Trans. on Knowl. and Data Eng.*, 2014, to appear.

[15] A. S. Izchak Sharfman and D. Keren, "Shape sensitive geometric monitoring," in *Proc. PODS*, 2008.

[16] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," *TODS*, 2007.

[17] G. Cormode, "The continuous distributed monitoring model," *SIGMOD Rec.*, 2013.

[18] B. Babcock and C. Olston, "Distributed top-k monitoring," in *Proc. SIGMOD*, 2003.

[19] N. Giatrakos, A. Deligiannakis, M. Garofalakis, I. Sharfman, and A. Schuster, "Prediction-based geometric monitoring over distributed data streams," in *Proc. SIGMOD*, 2012.

[20] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi, "Holistic aggregates in a networked world: distributed tracking of approximate quantiles," in *Proc. SIGMOD*, 2005.

[21] W. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space," in *Conference in Modern Analysis and Probability (New Haven, Conn., 1982)*, ser. Contemporary Mathematics, 1984.

[22] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *J. Comput. Syst. Sci.*, 1999.

[23] D. Achlioptas, "Database-friendly random projections: Johnson-lindenstrauss with binary coins," *J. Comput. Syst. Sci.*, 2003.

[24] J. Matoušek, "On variants of the Johnson–Lindenstrauss lemma," *Random Structures & Algorithms*, 2008.

[25] A. Bar-Or, D. Keren, A. Schuster, and R. Wolff, "Hierarchical decision tree induction in distributed genomic databases," *IEEE Trans. on Knowl. and Data Eng.*, 2005.

[26] R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication-efficient distributed monitoring of thresholded counts," in *Proc. SIGMOD*, 2006.

[27] J. Branch, B. Szymanski, C. Giannella, R. Wolff, and H. Kargupta, "In-network outlier detection in wireless sensor networks," in *Proc. ICDCS*, 2006.

[28] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, "Online outlier detection in sensor data using non-parametric models," in *Proc. VLDB*, 2006.

[29] N. Giatrakos, Y. Kotidis, A. Deligiannakis, V. Vassalos, and Y. Theodoridis, "TACO: tunable approximate computation of outliers in wireless sensor networks," in *Proc. SIGMOD*, 2010.

[30] L. Huang, X. Nguyen, M. Garofalakis, J. Hellerstein, M. Jordan, A. Joseph, and N. Taft, "Communication-efficient online detection of network-wide anomalies," in *Proc. INFOCOM*, 2007.

[31] S. Burdakis and A. Deligiannakis, "Detecting outliers in sensor networks using the geometric approach," in *Proc. ICDE*, 2012.

[32] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan, "Maintaining variance and k-medians over data stream windows," in *Proc. PODS*, 2003.

[33] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, "Optimal sampling from distributed streams," in *Proc. PODS*, 2010.

[34] I. Sharfman, A. Schuster, and D. Keren, "Aggregate threshold queries in sensor networks," in *IPDPS*, 2007.

[35] M. Gabel, D. Keren, and A. Schuster, "Communication-efficient outlier detection for scale-out systems," in *Proc. Workshop on Big Dynamic Distributed Data (BD3)*, 2013.