Communication-efficient Outlier Detection for Scale-out Systems

Moshe Gabel
Technion
Haifa, Israel
mgabel@cs.technion.ac.il

Daniel Keren
Haifa University
Haifa, Israel
dkeren@cs.haifa.ac.il

Assaf Schuster
Technion
Haifa, Israel
assaf@cs.technion.ac.il

ABSTRACT

Modern scale-out services are built on top of large datacenters composed of thousands of individual machines. These must be continuously monitored because unexpected failures can overload fail-over mechanism and cause large-scale outages. Such monitoring can be accomplished by periodically measuring hundreds of performance metrics and looking for outliers, often caused by misconfigurations, hardware failures or even software bugs. Previous work has shown that many failures are indeed preceded by such performance outliers, known as *performance problems* or *latent faults*.

In this work we adapt an existing unsupervised statistical framework for latent fault detection to provide an online, communication- and computation-reduced version. The existing framework is effective in predicting machine failures days before they happen, but requires each monitored machine to send all its periodic metric measurements, which is prohibitive in some settings and requires that the datacenter provide parallel storage and processing. Our adapted framework is able to reduce the amount of data sent and the processing cost at the central coordinator by processing the data in situ, making it usable in wider settings.

We utilize techniques from the domain of stream processing, specifically *sketching* and *safe zones*, to trade-off accuracy for communication and computation, without compromising its advantages. Like the original framework, our adapted framework is unsupervised, does not require domain knowledge, and provides statistical guarantees on the rate of false positives. Initial experiments show that scores yielded by the adapted framework match the original scores very well, while reducing communications by over 90%.

1. INTRODUCTION

In recent years the demand for computing power and storage has increased. Modern Web services and clouds rely on large datacenters, often comprised of thousands of machines. For such large services, it is unreasonable to assume that all machines are working properly and are well configured.

Monitoring is essential in datacenters, since unnoticed faults might accumulate to the point where redundancy and fail-over mechanisms break. Yet the large number of machines in datacenters makes manual monitoring impractical. Instead machines are usually monitored by collecting and analyzing performance counters [3, 5, 11]. Hundreds of counters per machine are reported by the various service layers, from service-specific metrics (such as database query statistics) to general metrics (such as CPU utilization).

In this work we adapt an existing fault detection algorithm [9] using sketching [8, 18, 7] and safe zones [17, 21] to reduce communication and processing requirements by an order of magnitude, while preserving its advantages.

Many existing failure detectors are inflexible [9], and most require centralizing the data in some form. Rule-based failure detectors define a set of watchdogs [11] that monitor specific counters and trigger an alert whenever a predefined threshold is crossed. However, maintaining these static rules requires ongoing manual adjustments.

More advanced methods model service behavior from historical logs. Supervised machine learning approaches [3, 6, 20, 4] train detectors on historic annotated data. Others [5] analyze logs from periods from when the service is guaranteed to be healthy to extract model parameters. Such approaches are sensitive to deviations in workloads and changes in the monitored service itself [23, 10]. After such changes the historical logs and the learned model are no longer relevant. Approaches that require labeled data can be expensive, since labels can be difficult to obtain, and re-labeling may be needed after service changes.

More flexible, unsupervised approaches have been proposed for high performance computing (HPC). Typical approaches [19, 22] analyze textual console logs to detect system or machine failures by examining frequency of log messages. Console logs are impractical in high-volume services for bandwidth and performance reasons: transactions are very short, time-sensitive, and rapid.

Finally, some approaches [14, 16] are unsupervised and flexible, but are not domain independent. They make use of domain insights and knowledge of the monitored service, for example in the domain of distributed file systems, and are therefore limited to specific systems.

Recent approaches to the monitoring problem [9, 16, 15] focus on early detection and handling of performance problems, or *latent faults*. These are outliers – machine behaviors that are indicative of a fault, or could eventually result in a fault, yet fly under the radar of monitoring systems because they are not acute enough, or were not anticipated by the

monitoring system designers. Early detection of latent faults can help prevent future failures and increase the reliability of services.

In previous work [9] we provided evidence that latent faults are common, and we 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 failures were preceded by latent faults. Furthermore, we were able to detect latent faults up to 14 days in advance of actual machine or software failures with up to 70% precision and 2% false positive rate – comparable to state of the art supervised techniques in controlled settings [4]. We demonstrated that our system is adaptable, requiring no domain knowledge, no labeled examples, and no parameter tuning in the face of workload changes and software updates. Finally, our system has proven and demonstrated guarantees on the false positive rates, it is non-intrusive, and it scales to very large services.

One drawback of previous work is the large communication and processing costs, prohibitive in some settings. Modern data centers are large, and consequently the resultant counter logs are also large. It may be very difficult to centralize and process such a large amount of data. In the experiments described in [9], the log files were over 10TB per day – too large to centralize and process in one location. Instead we relied on a data-parallel infrastructure [12] built into the data center. Parallel processing may not always be feasible in all situations, however. Furthermore, some large systems are not confined to a single datacenter but are geographically distributed.

In this work we extend our latent fault detection using techniques from the field of stream processing to reduce the size of the data by an order of magnitude, reducing communication and processing requirements, and allowing continuous online processing of distributed streams. The resulting technique is essentially a distributed outlier detector for multiple multivariate data streams, designed for monitoring large-scale online services.

2. SUMMARY OF PREVIOUS WORK

In [9] we presented a statistical latent fault detection framework with 3 derived tests. What follows is a short summary of that work, with the sign test as example.

2.1 Framework

We begin with a reasonable assumption: in a large cluster of machines doing the same job, most machines perform well most of the time. Further, we expect similar machines with similar hardware and software[1] to exhibit roughly similar behavior when measuring performance counters. We therefore compare these machines to find those whose performance differs notably.

There are M machines, each reporting C performance counters at every time t in a window of length T time points. We denote by $x(m,t)$ the vector of counter values for machine m at time t. The hypothesis is that the inspected machine is working properly and hence the statistical process that generated this vector for machine m is the same statistical process that generated the vector for any other machine m'. However, if we see that the vector $x(m,t)$ for machine m is notably different from the vectors of other machines, we reject the hypothesis and flag the machine m as *suspicious*, meaning we suspect it manifests a latent fault.

We now make explicit our assumptions on the behavior of the monitored machines: *a*) the majority of machines are working properly at any given point in time; *b*) the machines are homogeneous, meaning they perform a similar task and use similar hardware and software[2]; *c*) on average, the workload is balanced across all machines; *d*) the counters are ordinal and are reported at the same rate; and *e*) the counter values are memoryless in the sense that they depend only on the current time period (and are independent of the identity of the machine).

Formally, we assume that $x(m,t)$ is a realization of a random variable $X(t)$ whenever machine m is working properly. Since all machines perform the same task, and since the load balancer attempts to split the load evenly between the machines, the homogeneous assumption implies that we should expect $x(m,t)$ to show similar behavior. We do expect to see changes over time, due to changes in the workload, for example. However, we expect these changes to be similarly reflected in all machines.

At any time t, the input $x(t)$ to a test S consists of the vectors $x(m,t)$ for all machines m. The test $S(m,x(t))$ analyzes the data and assigns a *score* (either a scalar or a vector) to machine m at time t. Given a test S, and a significance level $\alpha > 0$, 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 longer the allowed time period for aggregating the scores is, the more sensitive the test will be. At the same time, aggregating over long periods of time creates latencies in the detection process. In our previous work we aggregated data over 24 hour intervals, as a compromise between sensitivity and latency.

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 [9] we derived and evaluated 3 different tests within the framework (different S functions). The sign test accumulates the average normalized direction from machine m to the rest of the machines. The Tukey test measures the average depth of $x(m,t)$ compared to the vectors of other machines at the same time. The LOF test similarly compares the local density of points around $x(m,t)$ to the local density of its neighbors. What follows is a summary of the sign test.

[1]These are reasonable assumptions in practice for many services and datacenters [14, 19].

[2]If this is not the case, we can often split the collection of machines to a few large homogeneous clusters.)

2.2 The Sign Test

The sign test extends the classic statistical sign test to allow the simultaneous 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 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, let $\mathcal{M}$ denote the set of all machines in a test, and $M = |\mathcal{M}|$ the number of machines. $\mathcal{T}$ are the time points where counters are sampled during preprocessing (for instance, every 5 minutes for 24 hours in our experiments), t denote a specific time point, and $T = |\mathcal{T}|$. Let m and m' be two machines and let $x(m,t)$ and $x(m',t)$ be the vectors of their reported and preprocessed counters at time t. We use the test

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

as a multivariate version of the sign function. If all the machines are working properly, we expect this value to be small. Therefore, the sum of several samples over time is also expected not to grow far from zero.

Algorithm 1: The sign test.

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

If all machines are working properly, 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 $p(m)$ in Algorithm 1 controls this statistic by guaranteeing a small number of false detections, depending on the significance level α.

3. ONLINE DETECTOR WITH REDUCED COMMUNICATION

We describe an online, communication-efficient version of the latent fault detector summarized in Section 2.

Detecting latent faults requires that each node must send all performance counters measured at each time point: T samples of C counters for each of the M machines. Beyond bandwidth costs, processing so much data is difficult to do on a single machine in a timely manner, due to the size and high dimensionality of the data. We apply two techniques to alleviate this issue.

Sketching is used to 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. The coordinator (or monitoring node) can then perform latent fault detection using the sketches, rather than the original data. In addition to reducing the communication load, this has the added benefit of reducing the computational load, since the dimensionality of the data is greatly reduced.

The framework in Section 2.1 requires that counter values be normalized during preprocessing (step 1), and this is true as well for the sketched version[3]. We use the safe zone approach [17] to monitor both the global mean and the global variance of each counter so that they do not deviate too much from their last known values. Each machine monitors whether its data satisfies a local constraint. If all local constraints at all machines are satisfied, the global mean and variance are known not to have deviated too far from their last known values. These last known values are then used to normalize the counter values at each node, before computing the sketch. If there is any violation, the coordinator polls each node for the current mean and variance, and distributes the new global mean and variance to all nodes.

The general pseudocode is shown in Algorithm 2 and explained in detail below.

3.1 Sketches

Sketching [18, 8] is a common technique used to process large, unpredictable data streams without having to send, store and process all data. It reduces the size of the data, while still enabling queries. See [7] for a recent survey of sketched-based (and other) distributed monitoring.

For our purposes, a *sketch* is a summary function that takes a vector and transforms it to a smaller vector while approximately preserving some desired property, for example inner products [1]. We use sketches to modify our tests to greatly reduce the amount of data that must be sent and processed. For example, 200 counters could be reduced to 10 dimensions, achieving an immediate 95% reduction in size.

Formally, rather than apply test S to the set of all local counter vectors $x(m,t)$, each machine m will first apply a sketching function f to its vectors, and send only the sketch $\hat{x} = f(x(m,t))$ for processing. The modified test $\hat{S}$ will be applied to the sketches rather than the original vector: $v_m = \frac{1}{T} \sum_t \hat{S}(m, \hat{x}(t))$.

One well-suited sketch is the AMS sketch [1], which involves a random linear projection to k dimensions. In our setting, each machine would project its counter vectors to k dimensions using a specially constructed projection matrix: $\hat{x}(m,t) = f(x(m,t)) = Rx(m,t)$ where R is a random $C \times k$ matrix constructed as described in [1].

The AMS sketch is general enough so that the same sketch can be used as input to different tests. Because the sign test relies on normalized directions, and since AMS sketches are linear projections, the sign test can be applied directly to the sketch. In other words, the sum of projected vectors is

[3]Automatic counter selection (part step 1) can be done in advance, offline, using the method described in [9].

Algorithm 2: Online detection pseudocode.

OFFLINE:
Automatically select counters.

INIT / COORDINATOR SYNC:
foreach *counter i in counters* **do**
  Poll all nodes for mean and variance of counter i.
  Distribute new global mean, variance, safe zones.
end

NODE at time point t:
foreach *counter i in counters* **do**
  if *counter not in safe zone* **then**
    Violation: send local mean, variance to coordinator.
    Wait for new global mean and variance.
  end
  Let x_i = value of counter i at time t .
  Normalize x_i with last known global mean and variance.
end
Let x = vector of normalized counter values.
Compute sketch of x and send to coordinator.

COORDINATOR at time point t:
if *violation for counter i* **then**
  Run SYNC.
end
Receive sketches from all nodes.
Compute test function S on received sketches.
Add most recent test function result to v_m.
Subtract least recent test function result from v_m.
Calculate p-value for all machines and issue warnings.

the same as projecting the sum of the vectors. The resulting vector is still small for healthy machines and large for outliers. The Tukey test described in our previous work already relies on a very similar technique, and has been shown to be very effective. The LOF test depends on the distance of pairs of points. In this case, the Johnson-Lindenstrauss lemma [13] guarantees that the projection to $k = O\left(\frac{\log M}{\epsilon^2}\right)$ preserves the distances within a factor of $1 \pm \epsilon$. Since our method averages T comparisons per day in the integration phase, we can further expect that in practice the error will be smaller.

3.1.1 Sign Test on Linear Sketches

The sign test function (1) from Section 2.2 depends only on the normalized direction from $x(m,t)$ to the other vectors. Let B be the unit sphere in C dimensions. Given the assumptions in Section 2.1, for healthy machines the normalized directions to other machines tend to be distributed spherically symmetric over B, resulting in the vector $v_m = \frac{1}{T}\sum_t S(m, x(t))$ being relatively short. Conversely, for machines with consistently anomalous behavior, v_m is a relatively long vector.

Given the sketched vectors $\hat{x}(m,t) = Rx(m,t)$, the sign test is still the sum of normalized directions from $x(m,t)$, after some transformation R. We now show that applying R to the unit sphere B maintains this symmetrical distribution. Let $R = UDV^T$ be the *singular value decomposition* of R.

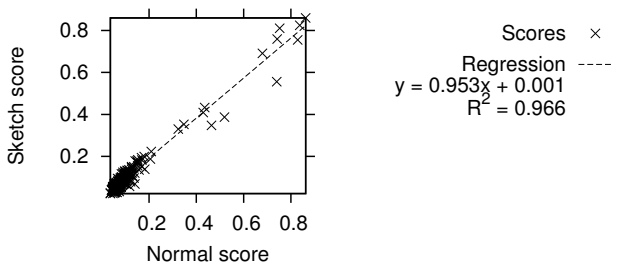


Figure 1: Sign test scores with AMS sketch compared to original scores. Sketch size is 8% of original data.

U and V^T are unitary matrices, and D is a diagonal matrix with positive elements. In geometrical terms, the transformation $R = UDV^T$ is a composition of rotation, followed by non-uniform scaling and dimensional reduction, and finally another rotation – all of which preserve the symmetric distribution around the origin. Therefore the transformation R maps the unit sphere B (in C dimensions) to an ellipsoid B' in k dimensions while preserving the symmetric distribution around the origin.

In summary, since the sign-test uses normalized directions and R preserves their symmetry around $x(m,t)$, we can apply the sign test directly to the sketched vectors $\hat{x}(m,t)$. Moreover, the sign test p-value does not depend on the dimensionality of the vectors, and so we can use it as is.

Preliminary experiments on counter logs from a small sample of 260 machines in a single day show that sign test scores and p-values computed on sketched data match the original very well. Figure 1 shows a comparison of sign test scores based on AMS sketches to regular (centralized, or parallel) sign test scores. The figure and linear regression show that the scores match very well, with $R^2 = 0.966$, very close to 1. The sketch reduced the data size by 92% – from 123 counters to 10 dimensions. The p-values are similarly close to the original values.

3.1.2 Online Integration Using a Sliding Window

The integration phase in stage 2 of the framework in Section 2.1 computes $v_m = \frac{1}{T}\sum_t S(m, x(t))$. Computing $S(m, x(t))$ only requires the data from time t, and therefore it is trivial to turn any test into an online test by keeping a window of test function (S) outputs for the last T sketches sent from the monitored machines. When new data arrives at time t, the coordinator updates the current v_m 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 for each machine in the time window can then be computed in the usual manner. Since the test function S need only be computed for the most recent time, and since the sketches are of low dimension k, processing and memory costs are low. This allows the computation to be done on a single coordinator machine on time, before the next round starts.

3.2 Scaling By Monitoring Variance

Our tests require the data to be standardized during preprocessing: each counter should be globally centered to zero mean and unit variance. In some settings we can assume that a counter's mean and variance do not change much,

or that they have a daily cycle. However, we might wish to avoid that assumption, and handle unpredictable workloads.

We use the *safe zones* approach [17, 21] to monitor both the global mean and the global variance of each counter. In this approach, each monitored machine receives a local constraint on its data $x(m,t)$ from a coordinator machine, such that if all local constraints are satisfied, the global monitored value $f(x(t))$ for some function f of the global aggregate is within a pre-defined threshold. Violations of local constraints are sent to the coordinator machine, which resolves them and sends updated local constraints to participating machines.

Given the last known global mean and variance of the last T samples, we define some lower and upper threshold, for example 0.9 and 1.1 times the last known values. If there is any violation, the coordinator polls each node for the current mean and variance, and distributes the new global mean and variance to all nodes. We can trade-off accuracy and communication by adjusting the high and low thresholds when monitoring. Violations are less likely if global mean and variance are allowed to drift further from their last known values – reducing communication but also decreasing accuracy [17].

We monitor each counter independently, so it is enough to show how we monitor a single counter X. Further note that all tests described in [9] are invariant to data translation, and so we do not monitor the global mean explicitly.

3.2.1 Notations

The set of values of counter X over the last T times and over M nodes (machines) is denoted by $X(t)$. We denote by $X_i(t)$ the values of X at node i for the last T times up to t. Thus $\mathrm{E}[X_i(t)]$ is the mean of the last T values at node i in time t, while $\mathrm{E}[X(t)]$ is the global mean of the last values at all nodes. Denote $\mu_i(t) = \mathrm{E}[X_i(t)]$ the local means, and $\mu(t) = \mathrm{E}[X(t)]$ the global mean. Similarly, we 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. Let $V(t) = (\mu(t), \lambda(t))$, and $V_i = (\mu_i(t), \lambda_i(t))$, the global and local monitored vectors, respectively.

3.2.2 Monitoring

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

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

We therefore monitor the conditions $L \leq \lambda - \mu^2 \leq H$, for some lower and upper variance thresholds L and H. Figure 2 shows the *admissible region* (the region in which the conditions hold), $0.5 \leq \lambda - \mu^2 \leq 1.5$. Following [17], we aim to find a convex safe zone G which is contained within the admissible region. 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 drift of 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 conditions on the drift vectors. The current local vectors can be written

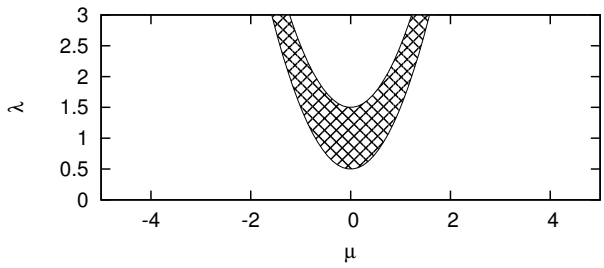


Figure 2: Admissible region for $L = 0.5, H = 1.5$.

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 therefore be written as the mean of drifts and the reference point:

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

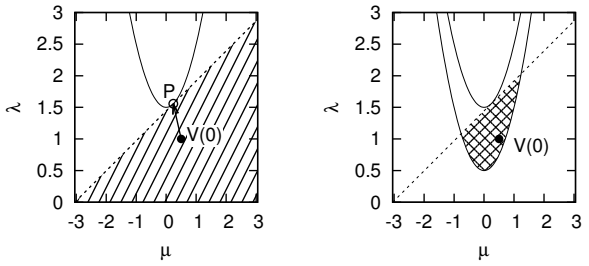
Let $W_i(t) = V(0) + d_i(t)$ be the local drift from the last reference point. Note that $V(t) = \frac{1}{M}\sum_i W_i$, recall G is convex, and from (2) we arrive at the local conditions: if $\forall i, W_i \in G$ then $V(t) \in G$.

To monitor that the variance is between L and H, we derive separate safe zones: one for variance above L and another for variance below H. As long as the local conditions for both safe zones are maintained 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 so that as long as $V(t) \in G_L$ then $\mathrm{Var}(X) \geq L$. 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 condition for each node i is trivial: $I_i(t) \in G_L$: $I_i(t) = V(0) + d_i(t) = (a, b)$ and monitor that $b - a^2 \geq L$.

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

We use the reference point $V(0)$ to find the optimal hyperplane. The thresholds H and L are reset during synchronization, so obviously $V(0) \in G$. We can choose any half-space I such that $V(0) \in I$, but to avoid future unnecessary synchronization we choose I such that $V(0)$ is far from the boundary of G. 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 condition is $W_i \in I$. We can find P numerically, or by 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 finally the induced safe zone I: the half-plane $\lambda - 0.474\mu < 1.443$. Figure 3(a) shows $V(0)$, P and the resulting safe zone, and Figure 3(b) shows the intersection with the safe zone for the lower limit $L = 0.5$.



(a) Upper threshold. **(b) Upper and lower.**

Figure 3: Safe zones for $L = 0.5, H = 1.5$ where $V(0) = (0.5, 1)$.

3.2.3 Handling Violations

If one of the local conditions $W_j \in G$ 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 $V_i(t)$ 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, where it is used for monitoring and scaling.

In terms of communication, our synchronizations are fairly inexpensive. Each node sends only two numbers per counter (μ and λ), rather than the entire time window of T samples. They also improve the accuracy of scaling, since nodes have fresh global mean and variance. There are safe zone techniques that allow partial synchronization for further communication reduction, for example by balancing a node with local violation with another node that has enough slack [2].

4. FUTURE WORK

This work uses sketching and safe zones to adapt the latent fault detector in [9] to a streaming setting, resulting in an online, communication-efficient outlier detector for common scale-out systems. Preliminary results show that the adapted detector obtains very similar results to those of the original latent fault detector for the sign test. Future work will concentrate on adapting additional tests, evaluating the detector on real-world systems, and exploring the communication-accuracy trade-off.

5. ACKNOWLEDGMENTS

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

6. REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 1999.

[2] D. Ben-David. Violation resolution in distributed stream networks. Master's thesis, Technion I.I.T, 2012.

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

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

[5] H. Chen, G. Jiang, and K. Yoshihira. Failure detection in large-scale internet services by principal subspace mapping. *IEEE Trans. Knowl. Data Eng.*, 2007.

[6] I. Cohen, M. Goldszmidt, T. Kelly, and J. Symons. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *Proc. OSDI*, 2004.

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

[8] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *SIGMOD*, 2007.

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

[10] C. Huang, I. Cohen, J. Symons, and T. Abdelzaher. Achieving scalable automated diagnosis of distributed systems performance problems. Technical report, HP Labs, 2007.

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

[12] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proc. EuroSys*, 2007.

[13] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, Contemporary Mathematics. 1984.

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

[15] 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.

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

[17] D. Keren, I. Sharfman, A. Schuster, and A. Livne. Shape sensitive geometric monitoring. *Knowledge and Data Engineering, IEEE Transactions on*, 2012.

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

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

[20] D. Pelleg, M. Ben-Yehuda, R. Harper, L. Spainhower, and T. Adeshiyan. Vigilant: out-of-band detection of failures in virtual machines. *SIGOPS Oper. Syst. Rev.*, 2008.

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

[22] 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.

[23] 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.