Latent Fault Detection With Unbalanced Workloads

Moshe Gabel
Technion – Israel Institute of Technology
Haifa 32000 Israel
mgabel@cs.technion.ac.il

Kento Sato*
Lawrence Livermore National Laboratory
Livermore, CA 94551 USA
kento@llnl.gov

Daniel Keren
Haifa University
Haifa 31905 Israel
dkeren@cs.haifa.ac.il

Satoshi Matsuoka
Tokyo Institute of Technology
Meguro-ku, Tokyo 152-8552
Japan
matsu@is.titech.ac.jp

Assaf Schuster
Technion – Israel Institute of Technology
Haifa 32000 Israel
assaf@cs.technion.ac.il

ABSTRACT

Big data means big datacenters, comprised of hundreds or thousands of machines. With so many machines, failures are commonplace. Failure detection is crucial: undetected failures may lead to data loss and outages.

Recent health monitoring approaches use anomaly detection to forecast failures – anomalous machines are considered to be at risk of future failures. Our previous work focused on detecting latent faults in large web services, which are often characterized by scale-out architecture where load is dynamically balanced. We proposed a robust and unsupervised latent fault detector for such systems, with statistical bounds on the rate of false positives. That detector, however, is unsuitable for applications without dynamic load balancing, such as statically-balanced key-value stores, Hadoop jobs, and supercomputer applications.

We describe an improved latent fault detection method for unbalanced workloads. It retains the advantages of our previous methods: it is unsupervised, robust to changes, and statistically sound. Moreover, the statistical bounds for the new method scale better with the number of machines, and so dramatically reduce the number of measurements needed. Preliminary evaluation on supercomputer logs shows that the new method is able to correctly predict some failures, while our previous methods completely fail in this setting.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Fault tolerance

Keywords

Anomaly detection, health monitoring, data mining

*This work was produced while K. Sato was at Tokyo Institute of Technology, Meguro-ku, Tokyo 152-8552 Japan.

1. INTRODUCTION

Recent years have seen an increasing demand for computing power and storage. Large scale applications – whether offline batch computations or modern web services and clouds – are implemented on top of large datacenters, comprised of thousands of machines or more. For large cloud services, it is unreasonable to assume that all machines are working properly and are well configured [29, 28]. Unnoticed failures may cause data loss and service outages. Similarly, modern supercomputers and high-performance clusters are increasingly comprised of more and more individual components (multiple CPUs, drives, and recently multiple GPUs [32]), resulting in higher failure rates [31, 1]. In such systems, failures may delay long computation, even to the point where little useful computation is being done [30, 31, 27].

Many current failure detectors model normal behavior from historical data. Modeling can be manual, by setting static thresholds [14], or automatic, using supervised machine learning [2]. Modeling from historical behavior is suboptimal, however [9]. Workload changes, data-dependent computations, and software updates render learned models inaccurate [12, 9]: static thresholds must be adjusted by domain experts, and machine learning models must be retrained from recent data. This retraining requires relabeling data as exhibiting normal and abnormal behavior – an expensive process. Furthermore, supervised techniques often only detect problems that have been foreseen or encountered before.

More recent approaches [17, 18, 19] focus on unsupervised methods (mainly anomaly detection) which require no labeling and less domain expertise. Within this context we focus on latent fault detection [9]. *Latent faults* are subtle behavior deviations that may indicate problems or misconfigurations. The aim is to catch unforeseen faults that "fly under the radar" of monitoring systems, before they manifest as machine or software failures. In our previous work [9] we proposed a statistical latent fault detection framework for web services. It is robust to software changes and workload fluctuations, and provides statistical bounds on the rate of false positives. Our evaluation showed that latent faults are common and can precede failures by days. We have also extended that work for distributed settings [8], where the goal is to reduce communication and computational load.

Despite its advantages, our existing latent fault detection framework, like many other anomaly detection techniques,

assumed that workload is dynamically distributed over identical machines. Though this setup is common in scale-out, replicated services, it is not always the case in every setting. First, some large-scale web services are statically balanced or simply poorly balanced. Consider, for example, a key-value store where keys are statically assigned to machines by a hash function. If commonly used keys fall on a small number of machines, these machines are much more heavily loaded than the rest. Similarly, parallel computation frameworks such as Hadoop generally use key values to partition loads, resulting in unbalanced workloads [22]. Finally, large scale computations in compute clusters may distribute work to nodes unequally, due to data locality or because there is no easy way to predict how data distribution affects computation.

Our previous work also required a large number of measurements for a single run of the detection algorithm. The statistical bound grew linearly weaker with the number of machines: the more machines, the larger the required time window.

This work proposes a statistical latent fault detection test for unbalanced workloads, making it more practical in settings such as supercomputers, and in other large scale systems whose computational workload is not necessarily balanced. It also reduces the window size from a full day (roughly 300 measurements) to minutes (4 measurements). Since the new bound scales much better as the number of monitored machines grows, the new detector is much more responsive to immediate changes. It can therefore be used to monitor large systems when rapid response is important.

A preliminary evaluation on historical metric and failure logs from the TSUBAME2 supercomputer[1] shows that the new detector is superior: while our previous latent fault detectors fail completely in this setting, the new detector can predict some failures several days in advance.

2. IMPROVED ANOMALY DETECTION

Our previous latent fault detection framework [9, 8] relied on several assumptions, which we now revisit. First, machines in the system are homogenous in terms of hardware, software infrastructure and running code. Second, the majority of machines are not faulty; in a large system, most machines perform well most of the time. Finally, the monitored system uses dynamic load balancing – on average, workload is distributed evenly across machines. Thus when measuring aggregated performance counters, we could expect healthy machines to exhibit the same behavior, on average. We wish to keep the first two assumptions, but avoid the third.

As with web services, machines in compute clusters are often homogenous for logistical reasons. Where they are not, it is often possible to cluster to a few distinct configurations, using hardware and job data. Indeed supercomputers have strict, almost uniform hardware. For example, TSUBAME2 has 6 types of nodes[2] with well-documented hardware configurations.

The second assumption is also quite reasonable; it is hard to imagine an expensive datacenter running for lengths of time with a majority of faulty machines.

We can no longer assume dynamic load balancing though.

[1] `http://www.gsic.titech.ac.jp/en/tsubame2` .

[2] `http://tsubame.gsic.titech.ac.jp/docs/guides/tsubame2/html_en/overview.html#computing-nodes` .

Table 1: Hypothetical machine measurements. Node D has anomalous memory and CPU usage.

Node	Reqs	Memory	DB	CPU
A	3	630	9	6
B	5	650	15	10
C	4	640	12	8
D	3	**740**	9	**15**
E	8	680	24	16
F	7	670	21	14
G	5	645	15	11

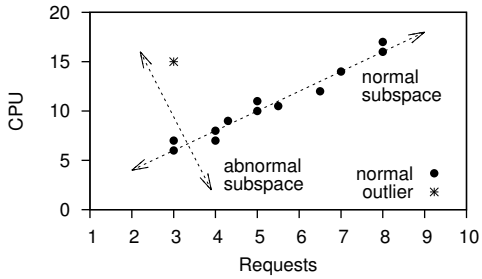


Figure 1: Scatterplot of the hypothetical requests vs CPU usage, with normal and abnormal subspaces. Machine D is visibly an outlier.

Instead, given our original assumption of homogenous machines running the same code, we assume there are common, inherent interdependencies, or correlations, between different counters, stemming from the fact that all machines do the same job, and run the same code. These correlations (not necessarily linear) between sets of counters are the result of what the machines are doing, and are not affected by workload. For instance, suppose task A requires k of some resource X and n of some resource Y for each unit of work. Increasing the workload to 5 units of work will require $5k$ of X and $5n$ of Y, but the correlation between k and n remains; it will remain even if we don't measure the actual workload. Generalizing to dependencies that may involve more than two counters, we can instead discuss correlations or relationships between or within sets of counters; for example, $2k + 3n + l^2 = m$.

Given the first two basic assumptions and the above, we can assume that similar machines doing the same task will result in the same correlations (relationships) between sets of counter values. The counters of machines with latent faults will not exhibit the same relationships.

For example, consider a hypothetical web service where for each client request we need 10MB of memory, 3 database transactions, and 2% CPU time. Machines with latent faults might have too few DB transactions, or too much memory use, or CPU usage that doesn't match the workload. Table 1 shows 5 such hypothetical machines. Machines A, B, C and E all exhibit the expected relationship between their counter values. Machine D, however, is anomalous, because its memory and CPU usage are far too high for its workload of 3 requests, for example due to a memory leak.

Our strategy is therefore to establish the linear correlations[3] within the aggregated counters at every time point,

[3] Not limited to the Pearson correlation, which is pairwise.

and find machines whose counters consistently (across several time points) exhibit different correlations.

2.1 PCA Subspace Decomposition

We will use Principal Component Analysis subspace decomposition [6, 23] to decompose the space of counters into a normal subspace and an abnormal subspace.

PCA is a statistical technique commonly used to automatically choose a smaller set of dimensions – the principal components – which captures most of the data variance. Since this subspace captures the variance in normal data, we can refer to it as the *normal subspace*. This normal subspace represents normal (healthy) linear correlations between the counters.

The residual components, on the other hand, define the complementary subspace that captures very little variance. In other words, when projecting a normal data point to the residual subspace, we can expect the projected vector to be very small, close to zero. The residual subspace is therefore the *abnormal subspace*, which represents violations of healthy relationships between counters.

Subsections 2.2 and 2.3 describe two variations using this basic idea. In the first variant, data vectors are projected into the abnormal subspace, and those vectors whose projection is above some threshold are declared to be abnormal. Jackson and Mudholkar [15] developed a way to infer the threshold from the data to guarantee a desired false positive rate. While their guarantee is only for multivariate Gaussian distributions, in practice the threshold is known to be robust even when the data is not Gaussian [16, 35]. Alternatively, we can apply the latent fault statistical framework [7] as is, using the projection to the abnormal subspace (normalized by the vector length) as the score function $S(m, x(t))$.

Figure 1 illustrates the technique. It shows requests vs. CPU usage of hypothetical machines from Table 1, including some additional healthy machines. The normal subspace is represented by the line $Y = 2X$, where X is requests and Y is CPU usage. The abnormal subspace is the perpendicular line. The outlier machine D clearly has a large presence in the abnormal subspace – projecting D's data to this space results in a large vector.

Ordinarily, historical data that is guaranteed to be "normal" is used to learn the normal and abnormal subspaces [23, 35]. In our case, we wish to detect small problems, and to support complex systems where we do not have a guaranteed error-free history. Instead we will make use of the large number of machines in the system. Since we assume most machines are fine at any given point in time, we can use this to extract the normal and abnormal subspaces.

One wrinkle in our plan is the presence of outliers in our data. Since we assume a small number of faulty machines (outliers), the resulting subspaces will include their faulty data. We therefore use a robust approach to PCA called HR-PCA, described by Xu et al. [34]. It is robust to outliers and arbitrarily corrupted data, and can recover the principal subspace even when the number of counters approaches the number of machines ($C \approx M$). HR-PCA is also quite efficient.

2.2 Formalizing the Algorithm

There are M machines, performing identical tasks, each periodically reporting C aggregated performance counters in a time window of length T. We standardize counter values in the time window across all machines to zero mean and unit variance in the time window. We denote by $x(m,t)$ the vector of standardized counter values for machine m at time t, and by $x(t) = \bigcup_m x(m,t)$ their union. Denote by X the $M \times C$ data matrix $x(t)$ after pre-processing and scaling at time t. Denote by x_m the row in X that came from machine m, meaning $x_m = x(m,t)$.

Using PCA we extract the normal subspace of X, comprised as the first k principal components $v_1, \ldots v_k$ that capture the most variance (say 95%). Denote by H_{no} the $C \times K$ normal subspace projection matrix built from the first k principal components, $H_{no} = [v_1, v_2, \ldots, v_k]$. Let the abnormal subspace projection matrix be the residual subspace $H_{ab} = I - H_{no}H_{no}^T$.

Given the projection H_{ab}, we can then map each machine vector $x(m,t)$ to its residual: $\tilde{x}_m = H_{ab}x_m$. Using the test statistic and threshold given in [15], define: $Q_m = \|\tilde{x}_m\|^2 = \|H_{ab}x_m\|^2$. A machine is declared abnormal at time t if $Q_m > Q_\alpha$, where Q_α denotes the threshold for the $1-\alpha$ confidence level:

$$Q_\alpha = \phi_1 \left[\frac{c_\alpha \sqrt{2\phi_2 h_0^2}}{\phi_1} + 1 + \frac{\phi_2 h_0 (h_0 - 1)}{\phi_1^2} \right]^{\frac{1}{h_0}} \quad ,$$

where

$$h_0 = 1 - \frac{2\phi_1\phi_3}{3\phi_2^2} \quad , \quad \phi_i = \sum_{j=k+1}^{C} \lambda_j^i \quad ,$$

λ_j is the variance captured by the j-th principal component, and c_α is the upper $1-\alpha$ percentile of the standard normal distribution. For a normal machine, $\Pr[Q_m > Q_\alpha] < \alpha$. In other words, α is the false alarm probability when testing a *single* machine.

Detecting one abnormal machine at time t is not sufficient, however. We are testing multiple machines, and must therefore guard against false positives. Hence we will only flag a machine if it is abnormal for T' consecutive times.

How big must T' be to guarantee a false alarm probability p when testing M machines? The probability of a false alarm for a specific machine m in T' consecutive time points is $\alpha^{T'}$. The false alarm probability in at least one machine after T' time points is therefore $1 - \left(1 - \alpha^{T'}\right)^M$, and so we require: $1 - \left(1 - \alpha^{T'}\right)^M \leq p$.

Thus for a desired false alarm probability p with M machines, we need a window size of:

$$T' = \left\lceil \log_\alpha \left(1 - \sqrt[M]{1-p}\right) \right\rceil . \tag{1}$$

Note that the probability of false alarms drops roughly exponentially with T'. We discuss this below in Subsection 2.5.

The final algorithm for target false probabilities p and α:

1. Preprocess: select counters and scale to unit variance.
2. For each time t across T' consecutive times:
   - 2.1 Compute robust PCA (HR-PCA) from data $x(t)$.
   - 2.2 Choose k that captures most variance (say 95%).
   - 2.3 Build H_{no}, H_{ab}, Q_α.
   - 2.4 For each machine m, check if $Q_m > Q_\alpha$.
3. Report m if $Q_m > Q_\alpha$ for T' consecutive times.

2.3 Alternative to Thresholding

The threshold Q_α is determined from the actual data, and so may be too conservative. It is possible that, due to noisy

data, the resulting threshold is too high. The test is binary: $Q_m > Q_\alpha$ is either true or false; there is no middle ground. Hence, it is possible that even if Q_m is consistently high, much higher than the Q of other machines, it is still below the threshold. Our conservative design to limit false positives will result in too many false negatives, as few faulty machines are flagged.

Instead, we can use the statistical framework from [9, 7]. 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 desired false alarm probability $0 < \alpha < 1$, 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.

We use the normalized Q_m as the score function:

$$S(m, x(t)) = \frac{Q_m}{\|x_m\|^2} = \frac{\|\tilde{x}_m\|^2}{\|x_m\|^2} = \frac{\|H_{ab}x_m\|^2}{\|x_m\|^2} \quad .$$

We derive probabilistic bounds using the machinery from [7]. Note that $0 \le S(m, x(t)) \le 1$, thus even if we change all of $x(t)$ S cannot change by more than 1. Moreover, HR-PCA is robust, so changing just one vector $x(m, t)$ should not overtly affect H_{ab}, thus $Q_{m'}$ for any other machine m' should not change. Therefore S is $1, 0$-*bounded* [7, Definition 2.3.1], and we can apply [7, Lemma 2.3.3] to get a p-value:

$$p(m) = (M+1)\exp\left(-\frac{2TM\gamma^2}{\left(\sqrt{M}+1\right)^2}\right) \qquad (2)$$

where $\gamma = \max(0, \|v_m\| - \hat{v})$, and $\hat{v} = \frac{1}{M}\sum_m \|v_m\|$. This p-value is the probability that $\|v_m\|$ is larger than the mean $\hat{v}$ by γ when m is healthy, given that we are testing m machines; testing for $p(m) < \alpha$ guarantees false alarm probability α across all machines, equivalent to p in Subsection 2.2.

$p(m)$ is more flexible than Q_α – it is computed from data of T times (step 2, above), rather than tested each time separately. Consistently small deviations from the norm accumulate if T is large enough. The advantage of this "soft threshold" approach is that it is much more sensitive to smaller anomalies ($Q < Q_\alpha$) than the original "all or nothing" approach. The downside is that the improved window size described in Subsection 2.5 no longer applies.

2.4 Unbalanced Workloads and Robustness

Our previous framework required that workload be dynamically balanced, on average, across all machines. This was because we directly compared counter values between machines. Aggregating across a large time window helped us overcome, and take advantage of, temporal noise. Indeed, temporary workload imbalance is very likely, since it is difficult to guarantee that all machines are equally loaded for any short time interval. Averaged across a larger interval, these small random imbalances cancel each other out.

Conversely, the algorithm described above does not depend on the absolute counter values, but instead on the correlations between them at each point in time. We expect that these correlations will remain similar regardless of the load.

Table 2: Hypothetical machine measurements exhibiting unbalanced load.

Load	Reqs	Memory	DB	CPU
low	8	680	24	16
low	5	650	15	10
low	6	660	18	12
high	33	930	99	66
high	40	1000	120	80
high	37	970	111	74

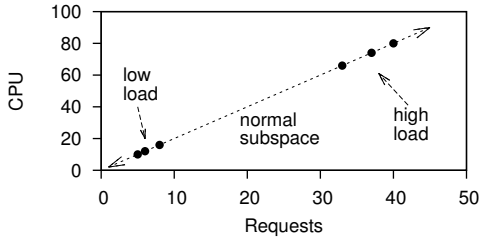


Figure 2: Scatterplot of the unbalanced hypothetical requests vs CPU usage. All machines lie on normal subspace.

For example, consider our hypothetical web service from above. For each client request, we need 10MB of memory, 3 database transactions, and 2% CPU time. Table 2 shows 6 such hypothetical machines. The first 3 machines are lightly loaded (few requests), while the last three are heavily loaded. Still, all exhibit the expected relationship between their counter values, and so lie on the normal subspace (Figure 2). A machine exhibiting anomalous CPU usage for the number of requests would lie outside this normal subspace.

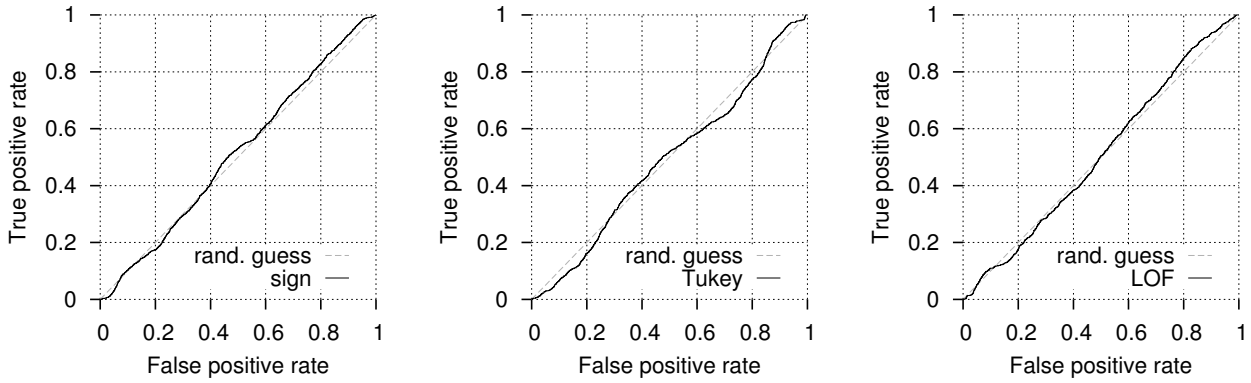
Moreover, as with our previous methods, the subspace decomposition approach is robust to changes in the monitored system. The normal and abnormal subspaces are recomputed using counter values measured at the same time, and we never compare such values across different times. If the software is updated, for example, the new behavior is never compared to the old one.

2.5 Improved Window Size

The bounds for our previous methods [9] required increasing window sizes as the number of machines grew. As illustrated by Equation (2), the framework bound grows linearly weaker with the number of machines M, meaning that we have to increase the window size T to compensate.

This can be intuitively be explained by the need to aggregate different measurements across many times to overcome temporal noise in counter values, such as short-term workload imbalance. The experiments described in [9] were performed with window size of $T = 288$, which translated to a full 24-hour day since counters were sampled every 5 minutes.

The PCA method has an improved window size. T' from Equation (1) is logarithmic in the number of machines M. The intuitive explanation is that we no longer need to track counter behavior across time to overcome minute random imbalances or random noise. For example, given $M = 10000$ machines, $\alpha = 0.01$, and overall false alarm probability $p = 0.01$, Equation (1) tells us we need a window size of only



(a) Sign test (b) Tukey test (c) LOF test

Figure 3: Performance of original latent fault detector on TSUBAME2 data with 7 day horizon.

3 time points, i.e., 15 minutes:

$$T' = \lceil \log_{0.01} \left(1 - \sqrt[10000]{1 - 0.01}\right) \rceil = \lceil 2.9989 \rceil = 3 \quad .$$

Even for a million machines, $M = 10^6$, a window size of 4 time points, meaning 20 minutes, is sufficient to guarantee a false positive rate of 0.01:

$$T' = \lceil \log_{0.01} \left(1 - \sqrt[10^6]{1 - 0.01}\right) \rceil = \lceil 3.9989 \rceil = 4 \quad .$$

3. PRELIMINARY EVALUATION

We used historical machine metric logs and failure records from the TSUBAME2 supercomputer to compare the new subspace decomposition approach to our existing latent fault detectors [9].

3.1 Supercomputer Workloads

Supercomputer workloads are very different from cloud workloads in many ways: long jobs rather than short requests; parallelization and load balancing are done within single jobs, not over all requests; and jobs are heterogeneous, so different nodes do not run the same code at the same time. Thus, the job uniformity assumption we make in Section 2 and our previous work [9] may no longer hold.

Computational jobs often perform many iterations of the same basic loop [30, 31]. For computations whose performance is not data-dependant (such as many common matrix operations), a single computation iteration will usually require the same amount of resources (CPU time, GPU load, etc.) as any other iteration. Moreover, required resources for one iteration will be the same for all nodes in the system with the same hardware configuration. This essentially brings us back to the PCA approach suggested in Section 2 – metrics of healthy nodes will lie in the same subspace.

Latent fault detection tests should be run on groups of machines partitioned by job. Scheduling logs that contain start and end times (along with the list of assigned machines) can be used to subdivide machines in this way.

3.2 The TSUBAME2 Dataset

We used 45 common machine metrics (e.g., cpu idle time, GPU utilization, user time, swap free, various temperatures), sampled every 1 to 10 minutes (depending on the metric), from one month of runs (roughly jobs, see below). We divided each run of 240 minutes or more, with at least 10 machines, to windows of length 240 minutes each. This resulted in 60 runs with a total of 252 windows, summarized in Table 3. We performed latent fault detection on each such window. The results of the detector were compared with the historical failure log within a 7 day horizon – a node is considered to have failed if it failed within 7 days from the time window; otherwise it is considered to have not failed.

Table 3: Statistics of inferred jobs (runs).

Statistic	Median	Max
Number of machines (M)	99	236
Length (minutes)	1016	5699

Since our TSUBAME2 logs did not include any job scheduling information, our preliminary experiments relied on CPU and GPU usage metrics to infer which machines were being used and how they were grouped. A group of machines that together became busy and then idle were considered to be a single job, or a "run". This method is ad-hoc and inherently inaccurate. For example, a failing machine might stop at the beginning of the computation and so would never be considered part of the run, as it did not finish with the rest. Section 5 discusses a potentially more robust alternative.

3.3 Results

We first evaluated the performance of our existing latent fault detectors: the sign, LOF and Tukey tests [9], with $T = 240$ and $\alpha = 0.01$. Figure 3 shows the receiver operating characteristic (ROC) curves for our existing latent fault detection tests. Ignoring computed p-values, we swept the threshold for anomaly scores $\|v_m\|$ (as in Eq. (2)) across a range of values and drew the resulting false positive and false negative rates. The performance of all three previous tests is no better than a random guess, where the false positive and false negative rates are equal.

We repeated the tests with the PCA approach suggested in Section 2, using the "soft threshold" variation described in Subsection 2.3. We used $T = 240$ and $\alpha = 0.01$. k was selected to capture 95% of variance. Finally, HR-PCA [34] was used as the robust PCA building block, with the maximum number of corrupted points set to 10% of machines

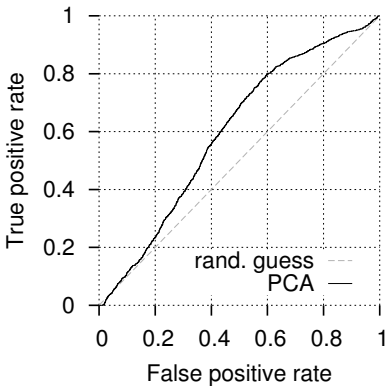


Figure 4: Performance of PCA latent fault detector on TSUBAME2 data with 7 day horizon.

($\hat{t} = 0.9M$). As can be seen in Figure 4, the subspace decomposition approach performs better than our original latent fault detectors. Though still no better than random guess at lower false positive rates, it is still able to predict some node failures several days ahead.

4. RELATED WORK

PCA residuals have been used in the past for monitoring tasks [6]. Lakhina et al. [23] famously used this approach to detect network traffic anomalies. Like many similar approaches, they monitor the network as a whole, and do not attempt to localize it to a specific node. Furthermore, they rely on historical data that is guaranteed to be normal, and assume that the system is unchanged, again a common theme. Xu et al. [35] analyze program source code to parse console log messages and use principal component analysis to identify unusual message patterns based on their frequency. As with Lakhina's work, this technique relies on error-free history and relatively stable systems. Console logs also tend to contain different sorts of data, and are likely to catch different sorts of anomalies. Similarly, Chen et al. [3] localize failures in software components of a Java application; they propose an online algorithm to update normal and abnormal behavior models. Chen et al. [4] also analyze the correlation between sets of measurements and track them over time. Their approach requires domain knowledge for choosing counters, and requires training on "healthy" periods to model baseline correlations.

Ling et al. [13] use Stochastic Matrix Perturbation theory to adapt Lakhina's work to distributed monitoring with PCA. Liu et al. [26], in turn, apply the distributed PCA monitoring approach on linear sketches of the network data to reduce running time and space costs.

5. CONCLUSIONS AND FUTURE WORK

Failure detection and prediction techniques are increasingly important in the era of clouds and compute clusters. Several recent approaches rely on anomaly detection techniques to detect failures ahead of time, while avoiding costly relabeling and retraining of models.

In this work we have presented a new latent fault detector suitable for settings where the workload is unbalanced. As with our previous methods, the new approach is robust to changes in the monitored system, it requires neither domain expertise nor labeled data, and it comes with statistical guarantees on the rate of false positives. Our preliminary evaluation showed that the new approach is clearly superior to the previous latent fault detectors in the supercomputer setting. Though the results obtained may not yet be practical in the supercomputer setting (possibly due to lack of scheduling logs), they show that the new detector does cope with unbalanced loads.

There are several avenues to pursue: more complete evaluation, communication-efficient and computation-efficient algorithms, and subspace clustering for job detection.

First, we wish to evaluate the new approach in additional settings where the workload is unbalanced. In the cloud setting, key-value stores are good candidates for our monitoring approach. In the compute cluster setting, Hadoop jobs have many machines running the same code in the reduce phase, but their computational load may be different.

Second, we can further combine PCA with distributed online detection as in [8], since collecting metrics from all nodes and computing PCA may be prohibitive for some large systems. We previously described [8] a communication-efficient approach using safe zones [21, 20] to standardize counter values across machines. Ling et al. [13] adapt PCA anomaly detection for distributed stream monitoring. We can combine their technique with recent distributed monitoring approaches [24, 11]. Beyond that, we can follow Liu et al. [26], who apply the distributed PCA technique on a linear sketch. Recent work by Liberty [25] introduces a better matrix sketching technique called Frequent Directions with improved bounds that is well-suited for PCA computation in a streaming setting. Indeed, streaming constructions of PCA using this approach are proposed by Ghashami and Phillips [10] and Cohen et al. [5].

Finally, more complete job and scheduling information in the supercomputer setting may help us improve results even further. A more robust approach to job detection might be *subspace clustering* [33]. Given a collection of vectors drawn from a union of (potentially disjoint) subspaces, subspace clustering algorithms cluster these data points according to the subspace they originate from. For similar reasons described in Section 2, and since hardware is uniform, we could assume that metrics of machines running the same job will lie in the same subspace. Thus, we might be able to use subspace clustering to identify machines that run similar code. Given such a group of machines, our anomaly detection techniques can be more effective. Moreover, the ability to cluster running code can be useful in monitoring settings such as virtual machine clouds, where operators have less information on what code runs inside virtual machines.

6. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme FP7-ICT-2013-11 under grant agreement Nº 619491 and Nº 619435. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-665287). This work was also supported by Grant-in-Aid for Research Fellow of the Japan Society for the Promotion of Science (JSPS Fellows) 24008253, and Grant-in-Aid for Scientific Research S 23220003.

7. REFERENCES

[1] G. Bronevetsky, I. Laguna, S. Bagchi, B. R. de Supinski, D. H. Ahn, and M. Schulz. Statistical fault detection for parallel applications with AutomaDeD. In *Proc. SELSE*, 2010.

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

[3] H. Chen, G. Jiang, C. Ungureanu, and K. Yoshihira. Failure detection and localization in component based systems by online tracking. In *Proc. KDD*, 2005.

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

[5] M. B. Cohen, S. Elder, C. Musco, C. Musco, and M. Persu. Dimensionality reduction for k-means clustering and low rank approximation. *CoRR*, 2014.

[6] R. Dunia and S. J. Qin. Multi-dimensional fault diagnosis using a subspace approach. In *Proc. ACC*, 1997.

[7] M. Gabel. Unsupervised anomaly detection in large datacenters. Master's thesis, Technion I.I.T, 2013.

[8] M. Gabel, D. Keren, and A. Schuster. Communication-efficient distributed variance monitoring and outlier detection for multivariate time series. In *Proc. IPDPS*, 2014.

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

[10] M. Ghashami and J. M. Phillips. Relative errors for deterministic low-rank matrix approximations. In *Proc. SODA*. 2014.

[11] N. Giatrakos, A. Deligiannakis, M. Garofalakis, I. Sharfman, and A. Schuster. Distributed geometric query monitoring using prediction models. *ACM Trans. Database Syst.*, 2014.

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

[13] 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.

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

[15] J. E. Jackson and G. S. Mudholkar. Control procedures for residuals associated with principal component analysis. *Technometrics*, 1979.

[16] D. R. Jensen and H. Solomon. A gaussian approximation to the distribution of a definite quadratic form. *Journal of the American Statistical Association*, 1972.

[17] S. Kadirvel, J. Ho, and J. A. B. Fortes. Fault management in Map-Reduce through early detection of anomalous nodes. In *Proc. ICAC*, 2013.

[18] 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.

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

[20] 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.

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

[22] Y. Kwon, K. Ren, M. Balazinska, and B. Howe. Managing skew in Hadoop. *IEEE Data Eng. Bull.*, 2013.

[23] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Proc. SIGCOMM*, 2004.

[24] A. Lazerson, I. Sharfman, D. Keren, A. Schuster, M. Garofalakis, and V. Samoladas. Monitoring distributed streams using convex decompositions. In *Proc. VLDB*, 2015. To appear.

[25] E. Liberty. Simple and deterministic matrix sketching. In *Proc. KDD*, 2013.

[26] Y. Liu, L. Zhang, and Y. Guan. A distributed data streaming algorithm for network-wide traffic anomaly detection. *SIGMETRICS*, 2009.

[27] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proc. SC*, 2010.

[28] E. B. Nightingale, J. R. Douceur, and V. Orgovan. Cycles, cells and platters: An empirical analysis of hardware failures on a million consumer pcs. In *Proc. EuroSys*, 2011.

[29] N. Palatin, A. Leizarowitz, A. Schuster, and R. Wolff. Mining for misconfigured machines in grid systems. In *Proc. SIGKDD*, 2006.

[30] K. Sato, N. Maruyama, K. Mohror, A. Moody, T. Gamblin, B. R. de Supinski, and S. Matsuoka. Design and modeling of a non-blocking checkpointing system. In *Proc. SC*, 2012.

[31] K. Sato, A. Moody, K. Mohror, T. Gamblin, B. R. d. Supinski, N. Maruyama, and S. Matsuoka. FMI: Fault tolerant messaging interface for fast and transparent recovery. In *Proc. IPDPS*, 2014.

[32] U. Verner, A. Schuster, and M. Silberstein. Processing data streams with hard real-time constraints on heterogeneous systems. In *Proc. ICS*, 2011.

[33] R. Vidal. Subspace clustering. *IEEE Signal Processing Magazine*, 2011.

[34] H. Xu, C. Caramanis, and S. Mannor. Outlier-robust PCA: The high-dimensional case. *IEEE Transactions on Information Theory*, 2013.

[35] 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.