Latent Fault Detection in Large Scale Services

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

Ran-Gilad Bachrach, Nikolaj Bjørner
Microsoft Research
Microsoft
Redmond, WA, USA
{rang,nbjorner}@microsoft.com

***Abstract*—Unexpected machine failures, with their resulting service outages and data loss, pose challenges to datacenter management. Existing failure detection techniques rely on domain knowledge, precious (often unavailable) training data, textual console logs, or intrusive service modifications.**

We hypothesize that many machine failures are not a result of abrupt changes but rather a result of a long period of degraded performance. This is confirmed in our experiments, in which over 20% of machine failures were preceded by such *latent faults*.

We propose a proactive approach for failure prevention. We present a novel framework for statistical latent fault detection using only ordinary machine counters collected as standard practice. We demonstrate three detection methods within this framework. Derived tests are domain-independent and unsupervised, require neither background information nor tuning, and scale to very large services. We prove strong guarantees on the false positive rates of our tests.

***Index Terms*—fault detection; web services; statistical analysis; distributed computing; statistical learning**

I. Introduction

For large scale services comprising thousands of computers, it is unreasonable to assume that so many machines are working properly and are well configured [1], [2]. Unnoticed faults might accumulate to the point where redundancy and fail-over mechanisms break. Therefore, early detection and handling of *latent faults* is essential for preventing failures and increasing the reliability of these services.

A *latent fault* is machine behavior that is indicative of a fault, or could eventually result in a fault. This work provides evidence that latent faults are common. We show that these faults can be detected using domain independent techniques, and with high precision. This enables a proactive approach [3]: machine failures can be predicted and handled effectively and automatically without service outages or data loss.

Machines are usually monitored by collecting and analyzing performance counters [4], [5], [6], [7]. Hundreds of counters per machine are reported by the various service layers, from service-specific information (such as the number of queries for a database) to general information (such as CPU usage). The large number of machines and counters in datacenters makes manual monitoring impractical.

Existing automated techniques for detecting failures are mostly rule-based. A set of watchdogs [6], [3] is defined. In most cases, a watchdog monitors a single counter on a single machine or service: the temperature of the CPU or free disk space, for example. Whenever a predefined threshold is crossed, an action is triggered. These actions range from notifying the system operator to automatic recovery attempts.

Rule-based failure detection suffers from several key problems. Thresholds must be made low enough that faults will not go unnoticed. At the same time they should be set high enough to avoid spurious detections. However, since the workload changes over time, no fixed threshold is adequate. Moreover, different services, or even different versions of the same service, may have different operating points. Therefore, maintaining the rules requires constant, manual adjustments, often done only after a "postmortem" examination.

Others have noticed the shortcomings of these rule-based approaches. [8], [9] proposed training a detector on historic annotated data. However, such approaches fall short due to the difficulty in obtaining this data, as well as the sensitivity of these approaches to deviations in workloads and changes in the service itself. Others proposed injecting code into the monitored service to periodically examine it [1]. This approach is intrusive and hence prohibitive in many cases.

More flexible, unsupervised approaches to failure detection have been proposed for high performance computing (HPC). [10], [11], [12] analyze textual console logs to detect system or machine failures by examining occurrence of log messages. In this work, we focus on large scale online services. This setting differs from HPC in several key aspects. Console logs are impractical in high-volume services for bandwidth and performance reasons: transactions are very short, time-sensitive, and rapid. Thus, in many environments, nodes periodically report aggregates in numerical counters. Console log analysis fails in this setting: console logs are non-existent (or very limited), and periodically-reported aggregates exhibit no difference in rate for faulty, slow or misconfigured machines. Rather, it is their values that matter. Moreover, console logs originate at application code and hence expose software bugs. We are interested in counters collected from all layers to reveal both software and hardware problems.

The challenge in designing a latent fault detection mechanism is to make it agile enough to handle the variations in a service and the differences between services. It should also be non-intrusive yet correctly detect as many faults as possible with only a few false alarms. As far as we know, we are the first to propose a framework and methods that address all these issues simultaneously using aggregated numerical counters normally collected by datacenters.

Contribution

We focus on machine behavior that is indicative of a fault, or could eventually result in a fault. We call this behavior a *latent fault*. Not all machine failures are the outcome of latent faults. Power outages and malicious attacks, for instance, can occur instantaneously, with no visible machine-related warning. However, even our most conservative estimates show that at least 20% of machine failures have a long incubation period during which the machine is already deviating in its behavior but is not yet failing (Sec. IV-E).

We develop a domain independent framework for identifying latent faults (Sec. II). Our framework is unsupervised and non-intrusive, and requires no background information. Typically, a scalable service will use (a small number of) large collections of machines of similar hardware, configuration, and load. Consequently, the main idea in this work is to use standard numerical counter readings in order to compare similar machines performing similar tasks in the same time frames, similar to [13], [10]. A machine whose performance deviates from the others is flagged as suspicious.

To compare machines' behavior, the framework uses tests that take the counter readings as input. Any reasonable test can be plugged in, including non-statistical tests. We demonstrate three tests within the framework and provide strong theoretical guarantees on their false detection rates (Sec. III). We use those tests to demonstrate the merits of the framework on several production services of various sizes and natures, including large scale services, as well as a service that uses virtual machines (Sec .IV).

Our technique is agile: we demonstrate its ability to work efficiently on different services with no need for tuning, yet still guaranteeing false positive rate. Moreover, changes in the workload or even changes to the service itself do not affect its performance: in our experiments, suspicious machines that switched services and workloads remained suspicious.

The rest of the paper is organized as follows: We first describe the problem and out general framework in Sec. II. In Sec. III we use the framework to develop three specific tests. In Sec. IV we discuss our empirical evaluation on several production services. We survey related work in Sec. V, and finally summarize our results and their implication in Sec. VI.

II. Framework

Large-scale services are often made reliable and scalable by means of replication. That is, the service is replicated on multiple machines with a load balancing process that splits the workload. Therefore, similar to [13], [10], we expect all machines that perform the same role, using similar hardware and configuration, to exhibit similar behavior. Whenever we see a machine that consistently differs from the rest, we flag it as suspicious for a latent fault. As we show in our experiments, this procedure flags latent faults weeks before the actual failure occurs.

A. Framework Overview

To compare machine operation, we use *performance counters*. Machines in datacenters often periodically report and log a wide range of performance counters. These counters are collected from the hardware (e.g., temperature), the operating system (e.g., number of threads), the runtime system (e.g., garbage collected), and from application layers (e.g., transactions completed). Hundreds of counters are collected at each machine. More counters can be specified by the system administrator, or the application developer, at will. Our framework is intentionally agnostic: it assumes no domain knowledge, and treats all counters equally. Figure 5 shows several examples of such counters from several machines across a single day.

We model the problem as follows: there are M machines each reporting C performance counters at every time unit. We denote the *vector of counter values* for machine m at time t as $x(m,t)$. 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 for a latent fault. (Below we simply say the machine is *suspicious*.)

After some common preprocessing (see Section II-D), the framework incorporates pluggable *tests* (aka outlier detection methods) to compare machine operation. At any point t, the input $x(t)$ to the 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. x and x' denote sets of inputs $x(m,t)$ and $x'(m,t)$, respectively, for all m and t.

The framework generates a wrapper around the test, which guarantees its statistical performance. 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. Therefore, in our experiments, we have 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 Section II-E we present the general analysis used to compute the p-value from aggregated test scores.

Given a test S, and a significance level $\alpha > 0$, we can present the framework as follows:

1) Preprocess the data as described in Section II-D (can be done once, after collecting some data; see below);
2) Compute for every machine m the vector v_m =

$\frac{1}{T}\sum_t S(m, x(t))$ (integration phase);
3) Using the vectors v_m, compute p-values $p(m)$;
4) Report every machine with $p(m) < \alpha$ as suspicious.

To demonstrate the power of the framework, we describe three test implementations in Sec. III.

B. Notation

The cardinality of a set G is denoted by $|G|$, while for a scalar s, we use $|s|$ as the absolute value of s. The L_2 norm of a vector y is $\|y\|$, and $y \cdot y'$ is the inner product of y and y'. $\mathcal{M}$ denotes the set of all machines in a test, m, m', m^* denote specific machines, and $M = |\mathcal{M}|$ denotes the number of machines. $\mathcal{C}$ is the set of all counters selected by the preprocessing algorithm, c denotes a specific counter, and $C = |\mathcal{C}|$. $\mathcal{T}$ are the time points where counters are sampled during preprocessing (for instance, every 5 minutes for 24 hours in our experiments), t, t' denote specific time points, and $T = |\mathcal{T}|$.

C. Framework Assumptions

In modeling the problem we make several reasonable assumptions (see, e.g., [13], [10]) that we will now make explicit. While these assumptions might not hold in every environment, they do hold in many cases, including the setups considered in Section IV.

- The majority of machines are working properly at any given point in time.
- Machines are homogeneous, meaning they perform a similar task and use similar hardware and software. (If this is not the case, then we can often split the collection of machines to a few large homogeneous clusters.)
- On average, workload is balanced across all machines.
- Counters are ordinal and are reported at the same rate.
- 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.

D. Preprocessing

Clearly, our model is simplified, and in practice not all of its assumptions about counters hold. Thus the importance of the preprocessing algorithm: it eliminates artifacts, normalizes the data, and automatically discards counters that violate assumptions and hinder comparison. Since we do not assume any domain knowledge, preprocessing treats all counters similarly, regardless of type. Furthermore, preprocessing is fully automatic and is not tuned to the specific nature of the service analyzed.

Not all counters are reported at a fixed rate, and even periodic counters might have different periods. Non-periodic and infrequent counters hinder comparison because at any given time their values are usually unknown for most machines. They may also bias statistical tests. Such counters are typically event-driven, and have a different number of reports on different machines; hence they are automatically detected by looking at the variability of the reporting rate and are removed by the preprocessing.

Additionally, some counters violate the assumption of being memoryless. For example, a counter that reports the time since the last machine reboot cannot be considered memoryless. Such counters usually provide no insight into the correct or normal behavior because they exhibit different behavior on different machines. Consequently, preprocessing drops those counters. Automatic detection of such counters is performed similarly to the detection of event-driven counters, by looking at the variability of counter means across different machines.

The process of dropping counters is particularly important when monitoring virtual machines. It eliminates counters reflecting cross-talk between machines running in the same physical host. In our experiments, after the above filtering operations, we were typically left with more than one hundred useful counters (over two hundred in some systems); see Table IV.

Preprocessing also samples counters at equal time intervals (5 minutes in our implementation), so that machines can be compared at those time points. Finally, the counters are normalized to have a zero mean and a unit variance in order to eliminate artifacts of scaling and numerical instabilities.

There are many possible ways to measure variability. Our implementation is based on normalized median absolute deviation. The particulars are not critical to the framework, and were omitted for lack of space.

E. Framework Analysis

In this section we show how the p-values (step 3 in the framework) are computed. We use two methods to compute these values. The first method assumes the expected value of the scoring function is known when all machines work properly. In this case, we compare v_m to its expected value and flag machines that have significant deviations (recall that $v_m = \frac{1}{T}\sum_{t\in\mathcal{T}} S(m, x(t))$; see framework Step 2). The second method for computing the p-value is used when the expected value of the scoring function is not known. In this case, we use the empirical mean of $\|v_m\|$ and compare the values obtained for each of the machines to this value. Both methods take the number of machines M into account. The resulting p-values are the probability of one false positive or more across $\mathcal{T}$, regardless of the number of machines.

In order to prove the convergence of v_m, we use the L_1, L_2-bounded property of the test S, as follows:

Definition 1: A test S is L_1, L_2-bounded if the following two properties hold for any two input vector sets x and x', and for any m and t:

1) $\|S(m, x(t)) - S(m, x'(t))\| \le L_1$.

2) Let $\bar{x}$ be x where $x(m,t)$ is replaced with $x'(m',t)$. Then for any $m' \neq m$, $\|S(m,x(t)) - S(m,\bar{x}(t))\| \leq L_2$.

The above definition requires that the test is bounded in two aspects. First, even if we change all the inputs, a machine score cannot change by more than L_1. Moreover, if we change the counter values for a single machine, the score for any other machine cannot change by more than L_2.

The following lemmas define the two methods. Proofs are omitted due to lack of space.

Lemma 1: Consider a test S which is L_1, L_2- bounded. Assume that $\forall m,t, \quad x(m,t) \sim X(t)$. Then for every $\gamma > 0$,

$$\Pr\left[\exists m \text{ s.t. } \|v_m\| \geq E\left[\|v_m\|\right] + \gamma\right] \leq M \exp\left(-\frac{2T\gamma^2}{L_1^2}\right) \quad .$$

The lemma follows by applying the bounded differences inequality [14] and the union bound.

The next lemma applies to the case where the expected value of v_m is not known. In this case, we use the empirical mean as a proxy for the true expected value.

Lemma 2: Consider a test S that is L_1, L_2- bounded. Assume that $\forall m,t \quad x(m,t) \sim X(t)$, and that $\forall m,m'$, $E\left[\|v_m\|\right] = E\left[\|v_{m'}\|\right]$. Denote by $\hat{v} = {}^1\!/_M \sum_m \|v_m\|$. Then for every $\gamma > 0$,

$$\Pr\left[\exists m \text{ s.t. } \|v_m\| \geq \hat{v} + \gamma\right] \leq (M+1)\exp\left(-\frac{2TM\gamma^2}{\left(L_1\left(1+\sqrt{M}\right)+L_2(M-1)\right)^2}\right) \quad .$$

In the proof we show that the random variable $\hat{v}$ is a Lipschitz function and apply the bounded differences inequality [14] to $\hat{v}$. Finally, we obtain the stated result by combining with Lemma 1.

III. Methods

Using the general framework described in Sec. II, we describe three test implementations: the *sign test* (Sec. III-A), the *Tukey test* (Sec. III-B), and the *LOF test* (Sec. III-C). Their analyses provide examples of the use of the machinery developed in Section II-E. Other tests can be easily incorporated into our framework. Such tests could make use of more information, or be even more sensitive to the signals generated by latent faults. For many well-known statistical tests, the advantages of the framework will still hold: no tuning, no domain knowledge, no training, and no need for tagged data.

A. The Sign Test

The sign test [15] is a classic statistical test. It verifies the hypothesis that two samples share a common median. It has been extended to the multivariate case [16]. We extend it to allow the simultaneous comparison of multiple machines.

Let m and m' be two machines and let $x(m,t)$ and $x(m',t)$ be the vectors of their reported counters at time t. We use the test

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

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

Algorithm 1: The sign test. Output a list of suspicious machines with p-value below significance level α.

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

The following theorem shows that if all machines are working properly, the norm of v_m should not be much larger than its empirical mean.

Theorem 1: Assume that $\forall m \in \mathcal{M}$ and $\forall t \in \mathcal{T}, x(m,t)$ is sampled independently from $X(t)$. Let v_m and $\hat{v}$ be as in Algorithm 1. Then for every $\gamma > 0$,

$$\Pr\left[\exists m \in \mathcal{M} \text{ s.t. } \|v_m\| \geq \hat{v} + \gamma\right] \leq (M+1)\exp\left(\frac{-TM\gamma^2}{2\left(\sqrt{M}+2\right)^2}\right) \quad .$$

Proof: The sign test is $2, \frac{2}{M-1}$-bounded. Applying Lemma 2, we obtain the stated result. ∎

Theorem 1 proves the correctness of the p-values computed by the sign test. For an appropriate significance level α, Theorem 1 guarantees a small number of false detections.

A beneficial property of the sign test is that it also provides a fingerprint for the failure in suspicious machines. The vector v_m scores every counter. The test assigns high positive scores to counters on which the machine m has higher values than the rest of the population and negative scores to counters on which m's values are lower. This fingerprint can be used to identify recurring types of failures [4]. It can also be used as a starting point for root cause analysis, which is a subject for future research.

B. The Tukey Test

The Tukey test is based on a different statistical tool, the Tukey depth function [17]. Given a sample of points Z, the Tukey depth function gives high scores to points near the center of the sample and low scores to points near the perimeter. For a point z, it examines all possible half-spaces that contain the point z and counts the number of points of Z inside the half-space. The depth is defined as the minimum

number of points over all possible half-spaces. Formally, let Z be a set of points in the vector space R^d and $z \in R^d$; then the Tukey depth of z in Z is:

$$\mathsf{Depth}(z|Z) = \inf_{w \in R^d} \left(|\{z' \in Z \text{ s.t. } z \cdot w \leq z' \cdot w\}|\right) \quad .$$

Algorithm 2: The Tukey test. Output a list of suspicious machines with p-value below significance level α.

Let $I = 5$;
for $i \leftarrow 1, \ldots, I$ **do**
  $\pi_i \leftarrow$ random projection $R^C \rightarrow R^2$;
  foreach *time* $t \in \mathcal{T}$ **do**
    foreach *machine* $m \in \mathcal{M}$ **do**
      $d(i,m,t) \leftarrow \mathsf{Depth}\left(\pi_i(x(m,t))|x(t)\right)$;
    end
  end
end
foreach *machine* $m \in \mathcal{M}$ **do**
  $S(m,x(t)) \leftarrow \frac{2}{I(M-1)} \sum_i d\left(i,m,t\right)$;
  $v_m \leftarrow \frac{1}{T} \sum_t S\left(m,x(t)\right)$;
end
$\hat{v} \leftarrow \frac{1}{M} \sum_m v_m$;
foreach *machine* $m \in \mathcal{M}$ **do**
  $\gamma \leftarrow \max(0, \hat{v} - \|v_m\|)$;
  $p(m) \leftarrow (M+1) \exp\left(-\frac{2TM\gamma^2}{\left(\sqrt{M}+3\right)^2}\right)$;
  if $p(m) \leq \alpha$ **then**
    Report machine m as suspicious
  end
end

In our setting, we say that if the vectors $x(m,t)$ for a fixed machine m consistently have low depths at different time points t, then m is likely to be behaving differently than the rest of the machines.

However, there are two main obstacles in using the Tukey test. First, for each point in time, the size of the sample is exactly the number of machines M and the dimension is the number of available counters C. The dimension C can be larger than the number of points M and it is thus likely that all the points will be in a general position and have a depth of 1. Moreover, computing the Tukey depth in high dimension is computationally prohibitive [18]. Therefore, similarly to [19], we select a few random projections of the data to low dimension (R^2) and compute depths in the lower dimension.

We randomly select a projection from R^C to R^2 by creating a matrix $C \times 2$ such that each entry in the matrix is selected at random from a normal distribution. For each time t, we project $x(m,t)$ for all the machines $m \in \mathcal{M}$ to R^2 several times, using the selected projections, and compute depths in R^2 with a complexity of only $O\left(M \log(M)\right)$, to obtain the depth $d(i,m,t)$ for machine m at time t with the i'th projection. The score used in the Tukey test is the sum of the depths computed on the random projections:

$$S(m,x(t)) = \frac{2}{I\left(M-1\right)} \sum_i d\left(i,m,t\right) \quad .$$

If all machines behave correctly, v_m should be concentrated around its mean. However, if a machine m has a much lower score than the empirical mean, this machine is flagged as suspicious. The following theorem shows how to compute p-values for the Tukey test.

Theorem 2: Assume that $\forall m \in \mathcal{M}$ and $\forall t \in \mathcal{T}, x(m,t)$ is sampled independently from $X(t)$. Let v_m and $\hat{v}$ be as in Algorithm 2. Then for every $\gamma > 0$,

$$\Pr\left[\exists m \in \mathcal{M} \text{ s.t. } v_m \leq \hat{v} - \gamma\right] \leq (M+1) \exp\left(\frac{-2TM\gamma^2}{\left(\sqrt{M}+3\right)^2}\right) \quad .$$

Proof: The Tukey test is $1, \frac{2}{M-1}$-bounded since $0 \leq d\left(i,m,t\right) \leq M-1$. Applying Lemma 2 with $-v_m$ and $-\hat{v}$, we obtain the stated result. ∎

C. The LOF Test

The LOF test is based on the *Local Outlier Factor* (LOF) algorithm [20], which is a popular outlier detection algorithm. The LOF test attempts to find outliers by looking at the density in local neighborhoods. Since the density of the sample may differ in different areas, isolated points in less populated areas are not necessarily outliers. The greater the LOF score is, the more suspicious the point is, but the precise value of the score has no particular meaning. Therefore, in the LOF test the scores are converted to ranks. The rank $r(m,x(t))$ is such that the machine with the lowest LOF score will have rank 0, the second lowest will have rank 1, and so on. If all machines are working properly, the rank $r(m,x(t))$ is distributed uniformly on $0, 1, \ldots, M-1$. Therefore, for healthy machines, the scoring function

$$S(m,x(t)) = \frac{2r(m,x(t))}{M-1} \tag{1}$$

has an expected value of 1. If the score is much higher, the machine is flagged as suspicious. The correctness of this approach is proven in the next theorem.

Theorem 3: Assume that $\forall m \in \mathcal{M}$ and $\forall t \in \mathcal{T}, x(m,t)$ is sampled independently from $X(t)$. Let v_m be as defined in Algorithm 3. Then for every $\gamma > 0$,

$$\Pr\left[\exists m \in \mathcal{M} \text{ s.t. } v_m \geq 1 + \gamma\right] \leq M \exp\left(-\frac{T\gamma^2}{2}\right) \quad .$$

Proof: The LOF test is $2, \frac{2}{M-1}$-bounded since $0 \leq r\left(m,x(t)\right) \leq M-1$. Moreover, under the assumption of this theorem, the expected value of the score is 1. Applying Lemma 1, we obtain the stated result. ∎

Algorithm 3: The LOF test. Output a list of suspicious machines with p-value below significance level α.

foreach *time* $t \in \mathcal{T}$ **do**
  $l(m,t) \leftarrow$ LOF of $x(m,t)$ in $x(t)$;
  foreach *machine* $m \in \mathcal{M}$ **do**
    $r(m, x(t)) \leftarrow$ rank of $l(m,t)$ in $\{l(m',t)\}_{m' \in M}$;
    $S(m, x(t)) \leftarrow \frac{2r(m,x(t))}{M-1}$;
  end
end
foreach *machine* $m \in \mathcal{M}$ **do**
  $v_m \leftarrow \frac{1}{T}\sum_t S\left(m, x(t)\right)$;
  $\gamma \leftarrow \max(0, v_m - 1)$;
  $p(m) \leftarrow M \exp\left(-\frac{T\gamma^2}{2}\right)$;
  if $p(m) \leq \alpha$ **then**
    Report machine m as suspicious
  end
end

TABLE I
SUMMARY OF TERMS.

Term	Description
Suspicious	machine flagged as having a latent fault
Failing	machine failed according to health signal
Healthy	machine healthy according to health signal
Precision	fraction of failing machines out of all suspicious = Pr(failing | suspicious)
Recall (TPR)	fraction of suspicious out of all failing machines = Pr(suspicious | failing)
False Positive Rate (FPR)	fraction of healthy machines out of all suspicious = Pr(suspicious | healthy)

IV. Empirical Evaluation

We conducted experiments on live, real-world, distributed services with different characteristics. The LG ("large") service consists of a large cluster ($\sim$ 4500 machines) that is a part of the index service of a large search engine (Bing). The PR ("primary") service runs on a mid-sized cluster ($\sim$ 300 machines) and provides information about previous user interactions for Bing. It holds a large key-value table and supports reading and writing to this table. The SE ("secondary") service is a hot backup for the PR service and is of similar size. It stores the same table as the PR service but supports only write requests. Its main goal is to provide hot swap for machines in the PR service in cases of failures. The VM ("virtual machine") service provides a mechanism to collect data about users' interactions with advertisements in a large portal. It stores this information for billing purposes. This service uses 15 virtual machines which share the same physical machine with other virtual machines. We tracked the LG, PR and SE services for 60 days and the VM service for 30 days. We chose periods in which these services did not experience any outage.

These services run on top of a data center management infrastructure for deployment of services, monitoring, automatic repair, and the like [6]. We use the automatic repair log to deduce information concerning the machines' health signals. This infrastructure also collects different performance counters from both the hardware and the running software, and handles storage, a common practice in such datacenters. Therefore our analysis incurs no overheard nor any changes to the monitored service.

Collected counters fall into a wide range of types: common OS counters such as the number of threads, memory and CPU usage, and paging; hardware counters such as disk write rate and network interface errors; and unique service application counters such as transaction latency, database merges, and query rate.

A. Protocol Used in the Experiments

We applied our methods to each service independently and in a daily cycle. That is, we collected counter values every 5 minutes during a 24-hour period and used them to flag suspicious machines using each of the tests. To avoid overfitting, parameters were tuned using the historical data of the SE service. In order to reduce the false alarm rate to a minimum, the significance level α was fixed at 0.01.

To evaluate test performance, we compared detected latent faults to machine health signals as reported by the infrastructure at a later date. Health alerts are raised according to rules for detecting software and hardware failures. Our hypothesis is that some latent faults will evolve over time into hard faults, which will be detected by this rule-based mechanism. Therefore, we checked the health signal of each machine in a follow-up period (*horizon*) of up to 14 days immediately following the day in which the machine was tested for a latent fault. We used existing health systems to verify the results of our latent fault detection framework. In some cases we used manual inspection of counters and audit logs.

Unfortunately, because of limited sensitivity and missing logs, health information is incomplete. Failing or malfunctioning machines that the current watchdog based implementation did not detect are considered by default to be healthy. Similarly, machines with unreported repair actions or without health logs are considered by default to be healthy. When flagged as suspicious by our tests, such machines would be considered false positives. Finally, not all machine failures have preceding latent faults, but to avoid any bias we include all logged health alerts in our evaluation, severely impacting recall, defined below (Section IV-E estimates the amount of latent faults). Therefore, the numbers we provide in our experiments are underestimations, or lower bounds on the true prevalence of latent faults.

In our evaluation, we refer to machines that were reported healthy during the follow-up horizon as *healthy*; other machines are referred to as *failing*. Machines that were flagged by a test are referred to as *suspicious*. Borrowing from the information retrieval literature [21], we use *precision* to measure the fraction of failing machines out of all suspicious machines and *recall* (also called *true positive rate*, or TPR) to measure the fraction of suspicious machines out of all failing machines. We also use the *false positive rate* (FPR) to

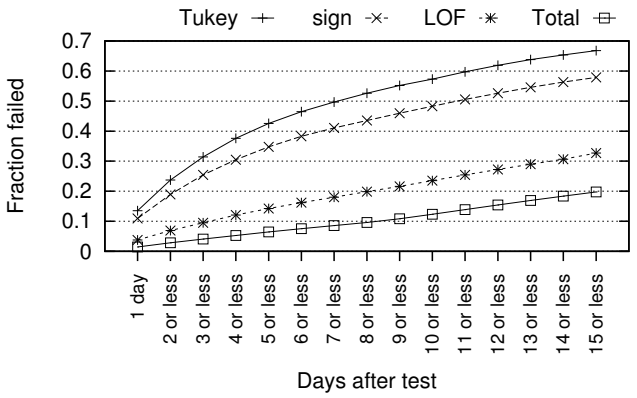


Fig. 1. Cumulative failures on LG service. Most of the faults detected by the sign test and the Tukey test become failures several days after detection.

denote the fraction of healthy machines out of all suspicious machines. Table I summarizes the terms used.

We applied the same techniques to all services, using the same choice of parameters. Yet, due to their different nature, we discuss the results for each service separately.

B. The LG Service

Table II shows a summary of the results (failure prediction) for the LG service. The low false positive rate (FPR) reflects our design choice to minimize false positives. Tracking the precision results proves that latent faults abound in the services. For example, the Tukey method has precision of $0.135, 0.497$ and 0.653 when failures are considered in horizons of $1, 7$ and 14 days ahead, respectively. Therefore, most of the machines flagged as suspicious by this method will indeed fail during the next two weeks. Moreover, most of these failures happen on the second day or later.

The recall numbers in Table II indicate that approximately 20% of the failures in the service were already manifested in the environment for about a week before they were detected.

The cumulative failure graph (Figure 1) depicts the fraction across all days of suspicious machines which failed up to n days after the detection of the latent fault. In other words, it shows the precision vs. prediction horizon. The "total" line is the fraction of all machines that failed, demonstrating the normal state of affairs in the LG service. This column is equivalent to a guessing "test" that randomly selects suspicious machines on the basis of the failure probability in the LG service. Once again, these graphs demonstrate the existence and prevalence of latent faults.

To explore the tradeoffs between recall, false positive rate, and precision, and to compare the different methods, we present *receiver operating characteristic* (ROC) curves and *precision-recall* (P-R) curves. The curves, shown in Figure 2, were generated by varying the significance level: for each value of α we plot the resulting false positive rate and true positive rate (recall) as a point on the ROC curve. The closer to the top-left corner (no false positives with perfect recall), the better the performance. A random guess would yield a diagonal line from $(0, 0)$ to $(1, 1)$. The P-R curve is similarly generated from recall and precision.

TABLE II
PREDICTION ON LG WITH SIGNIFICANCE LEVEL OF 0.01.

Period	Test	Recall	FPR	Precision
1 day	Tukey	0.240	**0.023**	**0.135**
	sign	**0.306**	0.037	0.109
	LOF	0.248	0.095	0.038
7 days	Tukey	0.151	**0.014**	**0.497**
	sign	**0.196**	0.026	0.411
	LOF	**0.203**	0.087	0.180
14 days	Tukey	0.093	**0.011**	**0.653**
	sign	0.126	0.022	0.563
	LOF	**0.162**	0.082	0.306

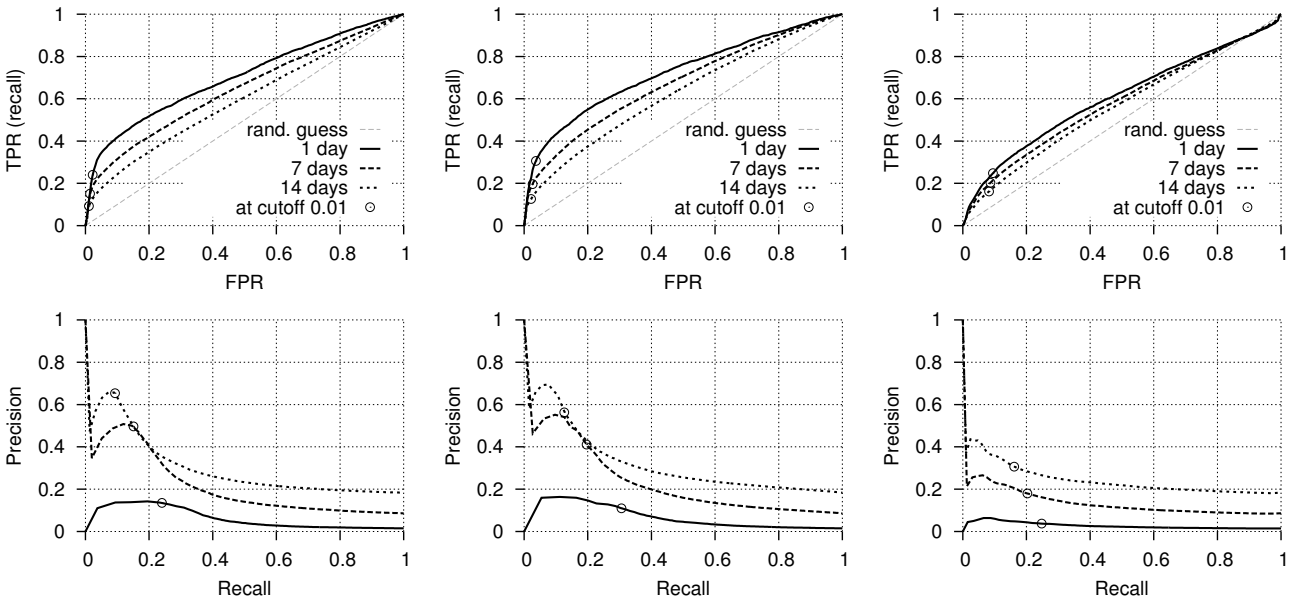
Both the Tukey and sign tests successfully predict failures up to 14 days in advance with a high degree of precision, with sign having a slight advantage. Both perform significantly better than the LOF test, which is still somewhat successful. The results reflect our design tradeoff: at significance level of 0.01, false positive rates are very low (around $2 - 3\%$ for Tukey and sign), and precision is relatively high (especially for longer horizons).

The dips in the beginning of the P-R curves reflect machines that consistently get low p-values, but do not fail. Our manual investigation of some of these machines shows that they can be divided into (1) undocumented failures (incomplete or unavailable logs), and (2) machines that are misconfigured or underperforming, but not failing outright since the services do not monitor for these conditions. Such machines are considered false positives, even though they are actually correctly flagged by our framework as suspicious. This is additional evidence that the numbers reported in our experiments are underestimates, and that latent faults go unnoticed in the environment. This is also why false positive rates are slightly higher than the significance level of 0.01.

Finally, we investigate the sensitivity of the different methods to temporal changes in the workload. Since this service is user facing, the workload changes significantly between weekdays and weekends. We plot Tukey prediction performance with a 14-day horizon for each calendar day (Figure 3). Note that the weekly cycle does not affect the test. The visible dips at around days 22, 35, and towards the end of the period, are due to service upgrades during these times. Since the machines are not upgraded simultaneously, the test detects any performance divergence of the different versions and reports these as failures. However, once the upgrade was completed, no tuning was necessary for the test to regain its performance.

C. PR and SE Services

The SE service mirrors data written to PR, but serves no read requests. Its machines are thus less loaded than PR machines, which serve both read and write requests. Hence, traditional rule-based monitoring systems are less likely to detect failures on these machines. The existence of latent faults on these machines is likely to be detected by the health mechanisms only when there is a failure in a primary machine,



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

Fig. 2. ROC and P-R curves on LG service. Highlighted points are for significance level $\alpha = 0.01$.

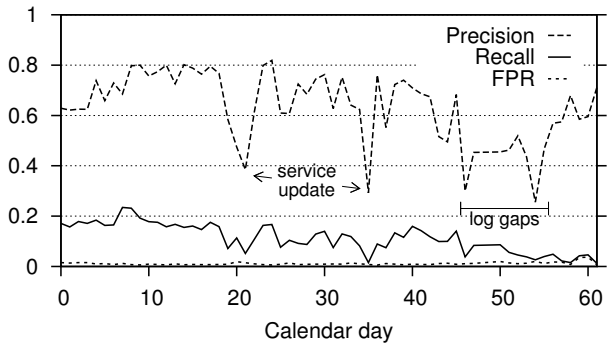


Fig. 3. Tukey performance on LG across 60 days, with 14-day horizon. It shows the test is not affected by changes in the workload, quickly recovering from service updates on days 22 and 35. Lower performance on days 45–55 is an artifact of gaps in counter logs and updates on later days.

followed by the faulty SE machine being converted to the primary (PR) role.

Unfortunately, the health monitors for the PR and SE services are not as comprehensive as the ones for the LG service. Since we use the health monitors as the objective signal against which we measure the performance of our tests, these measurements are less reliable. To compensate for that, we manually investigated some of the flagged machines. We are able to provide objective measurements for the SE service, as there are enough real failures which can be successfully predicted, despite at least 30% spurious failures in health logs (verified manually).

Performance on SE service for a significance level of 0.01 is summarized in Table III. ROC and P-R curves are in Figure 4. Our methods were able to detect and predict machine failures;

TABLE III
PREDICTION ON SE, 14-DAY HORIZON, SIGNIFICANCE LEVEL 0.01.

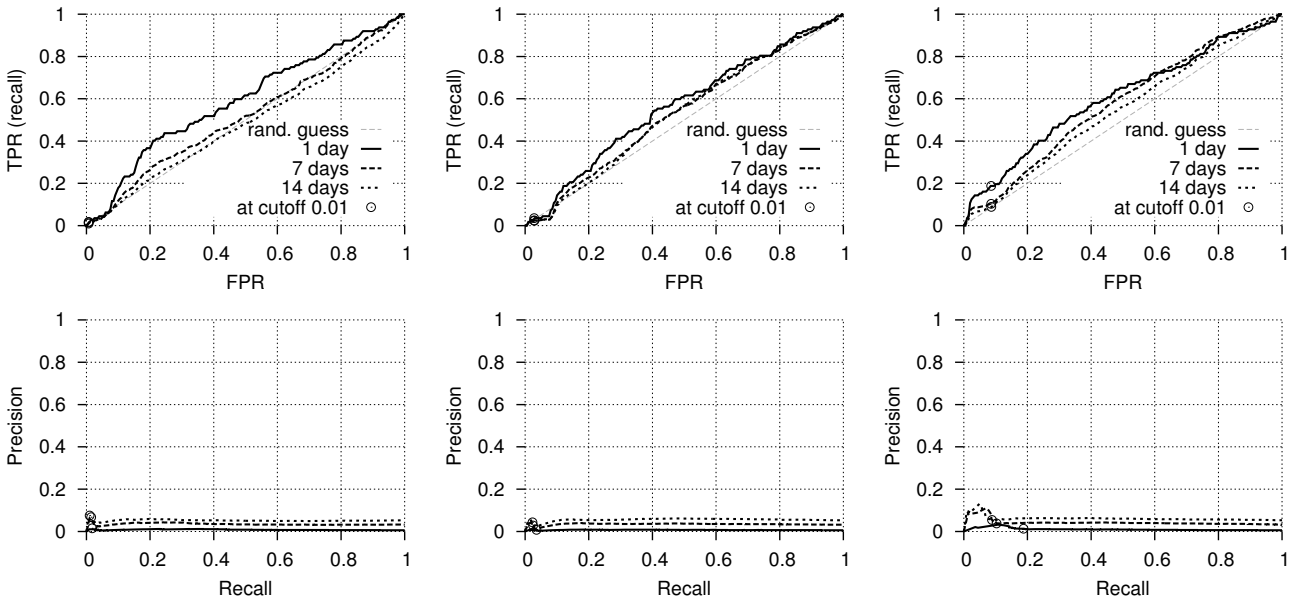
Test	Recall	FPR	Precision
Tukey	0.010	0.007	0.075
sign	0.023	0.029	0.044
LOF	0.089	0.087	0.054

therefore, latent faults do exist in this service as well, albeit to a lesser extent. As explained above, since this is a backup service, some of the failures go unreported to the service platform. Therefore, the true performance is likely to be better than shown.

The case of the PR service is similar to the SE service but even more acute. The number of reported failures is so low (0.26% machine failures per day) that it would be impossible to verify positive prediction. Nevertheless, all tests show very low FPR (about 1% for sign and Tukey, 7% for LOF), and in over 99% of healthy cases there were no latent faults according to all tests.

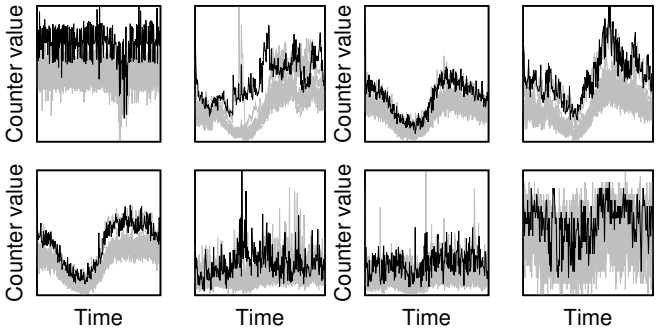
D. VM Service

The VM service presents a greater challenge, due to the use of virtual machines and the small machine population. In principle, a test may flag machines as suspicious because of some artifacts related to other virtual machines sharing the same host. Due to the small size of this cluster, we resort to manually examining warning logs, and examining the two machines with latent faults found by the sign test. One of the machines had high CPU usage, thread count, disk queue length and other counters that indicate a large workload, causing our test to flag it as suspicious. Indeed, two days after detection there was a watchdog warning indicating that the machine is



(a) Tukey (b) sign (c) LOF

Fig. 4. ROC and P-R curves on the SE service. Highlighted points are for significance level $\alpha = 0.01$.



(a) Detection day (b) Warning day

Fig. 5. Aberrant counters for suspicious VM machine (black) compared to the counters of 14 other machines (gray).

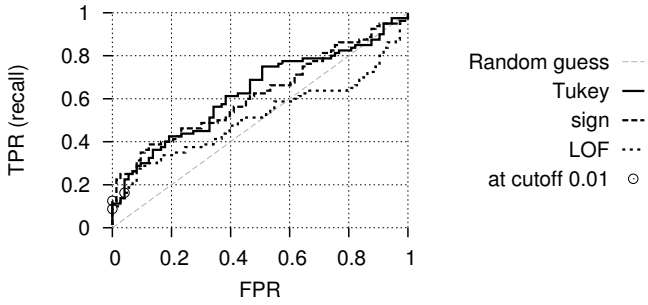


Fig. 6. Detection performance on LG service. At least 20-25% of failures have preceding latent faults. Highlighted points are for $\alpha = 0.01$.

overloaded. The relevant counters for this machine are plotted in Figure 5. The second machine for which a latent fault was detected appears to have had no relevant warning, but our tests did indicate that it had low memory usage, compared to other machines performing the same role.

E. Estimating the Number of Latent Faults

Some failures do not have a period in which they live undetected in the system. Examples include failures due to software upgrades and failures due to network service interruption. We conducted an experiment on the LG environment with the goal of estimating the percentage of failures which do have a latent period.

We selected 80 failure events at random and checked whether our methods detect them 24 hours before they are first reported by the existing failure detection mechanism. As a control, we also selected a random set of 73 machines known to be healthy. For both sets we require that events come from different machines, and from a range of times and dates.

For this experiment we define a failing machine to be a machine that is reported to be failing but did not have any failure report in the preceding 48 hours. We define a machine to be healthy if it did not have any failure during the 60 day period of our investigation. Figure 6 shows the ROC curves for this experiment. Failing machines where latent faults are detected are true positives. Healthy machines flagged as suspicious are counted as false positives. Both sign and Tukey manage to detect $20\% - 25\%$ of the failing machines with very few false positives. Therefore, we conclude that at least $20\% - 25\%$ of the failures are latent for a long period. Assuming our estimation is accurate, the recall achieved in Section IV-B is close to the maximum possible.

TABLE IV
AVERAGE NUMBER OF COUNTERS REMOVED. MANY COUNTERS REMAIN AFTER AUTOMATED FILTERING.

Counters	LG	VM	PR	SE
Event-driven	85	39	100	112
Slow	68	12	19	24
Constant	87	29	52	40
Varied means	103	30	57	79
Remaining	**211**	**106**	**313**	**89**
Total	554	216	541	344

F. Comparison of Tests

The three tests proposed in this work are based on different principles. Nevertheless, they tend to flag the same machines. For instance, more than 80% of the machines that were flagged by Tukey are also flagged by the sign test. All tests achieve a low false positive rate on all services, with the Tukey and sign tests matching the very low user-specified rate parameter.

To better characterize the sensitivities of the different tests, we evaluated them on artificially generated data to which we injected three types of "faults": counter location (offset), counter scale, or both. The strength of the difference varies across the failing machines, and we compare the sensitivity of each test to different kinds of faults. The resulting curves are shown in Figure 7. This experiment shows that the sign test is very sensitive to changes in offsets. LOF has some sensitivity to offset changes while the Tukey test has little sensitivity, if any, to this kind of change. When scale is changed, LOF is more sensitive in the range of low false positive rates but does not do well later on. Tukey is more sensitive than sign to scale changes.

G. Filtering Counters in Preprocessing

As described in Section II-D, the preprocessing stage removes some counters. Table IV reports the average number of counters removed in each service.

When removing counters violating the memoryless assumption, we measure the mean variability of each counter across all machines, leaving only counters with low variability. Our choice of a low fixed threshold value stems from our conservative design choice to avoid false positives, even at the price of removing potentially informative counters. Figure 8 justifies this choice: the majority of counters that were not filtered have relatively low variability on most services, whereas the higher variability range (2–10) typically contains few counters. Beyond 10 counters are not usable: most of them are effectively a unique constant value for this counter for each machine. Thus, tuning is not needed in preprocessing.

To further explore the effect of different thresholds, we measured the performance of the tests on a single day of the LG service with different mean variability thresholds. With strict significance level, higher thresholds result in slightly better recall but slightly lower precision, confirming our expectations.

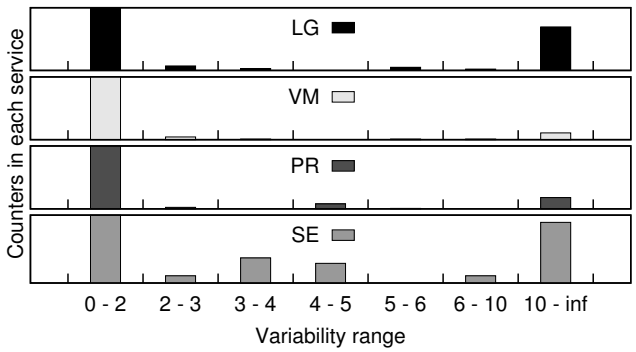
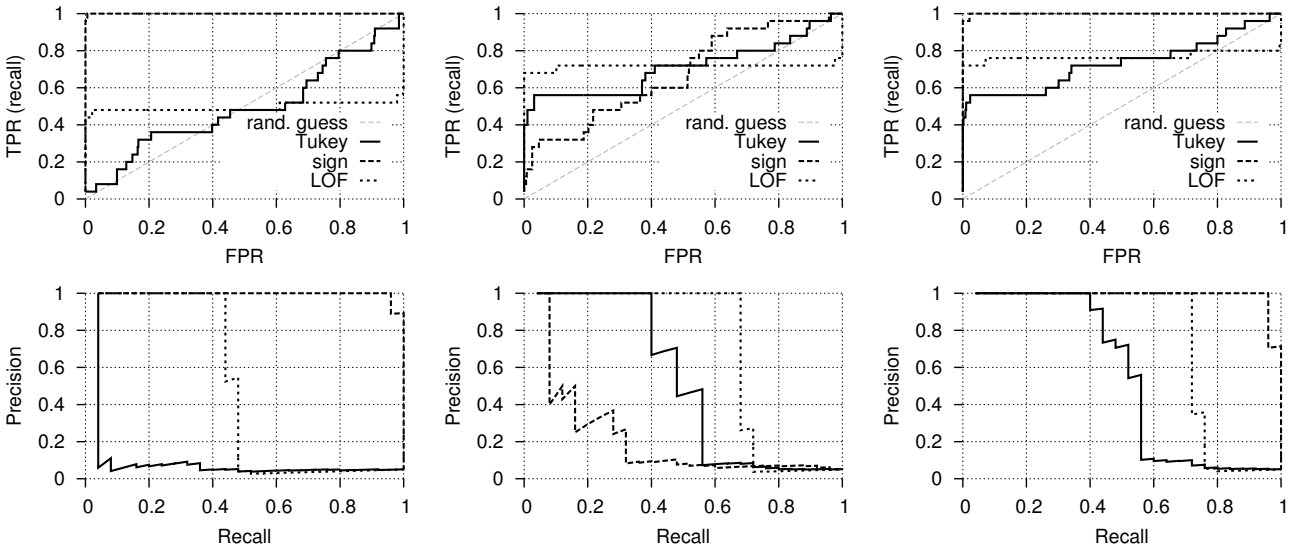


Fig. 8. Histogram of counter mean variability for all services. The majority of counters have variability below 2.

V. RELATED WORK

The problem of automatic machine failure detection was studied by several researchers in recent years, and proposed techniques have so far been mostly supervised, or reliant on textual console logs.

Chen et al. [5] analyze the correlation between sets of measurements and track them over time. This approach requires domain knowledge for choosing counters, and training to model baseline correlations. Chen et al. [8] presented a supervised approach based on learning decision trees. The system requires labeled examples of failures and domain knowledge. Moreover, supervised approaches are less adaptive to workload variations and to platform changes. Pelleg et al. [22] explore failure detection in virtual machines using decision trees. Though the basis is domain independent, the system is supervised, requiring training on labeled examples and manually selected counters. Bronevetsky et al. [23] monitor state transitions in MPI applications, and observe timing and probabilities of state transitions to build a statistical model. Their method requires no domain knowledge, but is limited to MPI-based applications and requires potentially intrusive monitoring. It also requires training on sample runs of the monitored application to achieve high accuracy. Sahoo et al. [7] compare three approaches to failure event prediction: rule-based, Bayesian network, and time series analysis. They successfully apply their methods to a 350-node cluster for a period of one year. Their methods are supervised and furthermore rely on substantial knowledge of the monitored system. Bodík et al. [4] produce fingerprints from aggregate counters that describe the state of the entire datacenter, and use these fingerprints to identify system crises. As with other supervised techniques, the approach requires labeled examples. The authors present quick detection of system failures that have already occured, whereas we focus on detection of latent faults ahead of machine failures. Cohen et al. [9] induce a tree-augmented Bayesian network classifier. Although this approach does not require domain knowledge other than a labeled training set, the classifier is sensitive to changing workloads. Ensembles of models are used in [24] to reduce the sensitivity of the former approach to workload changes,



(a) Location (b) Scale (c) Location+scale

Fig. 7. Performance on types of synthetic latent faults

at the cost of decreased accuracy when there are too many failure types ([25]).

Palatin et al. [1] propose sending benchmarks to servers in order to find execution outliers. Like our method, their approach is based on outlier detection, is unsupervised, and requires no domain knowledge. However, through our interaction with system architects we have learned that they consider this approach *intrusive*, because it requires sending jobs to be run on the monitored hosts, thus essentially modifying the running service. Kasick et al. [13] analyze selected counters using unsupervised histogram and threshold-based techniques. Their assumptions of homogenous platforms and workloads are also similar to ours. However they consider distributed file systems exclusively, relying on expert insight and carefully selected counters. Our technique requires no knowledge and works for all domains.

There are several unsupervised textual console log analysis methods. Oliner et al. [10] present Nodeinfo: an unsupervised method that detects anomalies in system messages by assuming, as we do, that similar machines produce similar logs. Xu et al. [12] analyze source code to parse console log messages and use principal component analysis to identify unusual message patterns. Lou et al. [11] represent code flow by identifying linear relationships in counts of console log messages. Unlike [10], [11], our method has strong statistical basis that can guarantee performance, and it requires no tuning. All three techniques focus on the unusual occurrences of textual messages, while our method focuses on numerical values of periodic events. Furthermore, we focus on early detection of latent faults in either hardware or software. Finally, console logs analysis is infeasible in large-scale services with high transaction volume.

VI. Conclusions

While current approaches focus on the identification of failures that have already occurred, latent faults manifest themselves as aberrations in some of the machines' counters, aberrations that will eventually lead to actual failure. Although our experiments show that latent faults are common even in well-managed datacenters, we are, as far as we know, the first to address this issue.

We introduce a novel framework for detecting latent faults that is agile enough to be used across different systems and to withstand changes over time. We proved guarantees on the false detection rates and evaluated our methods on several types of production services. Our methods were able to detect many latent faults days and even weeks ahead of rule-based watchdogs. We have shown that our approach is versatile; the same tests were able to detect faults in different environments without having to retrain or retune them. Our tests handle workload variations and service updates naturally and without intervention. Even services built on virtual machines are monitored successfully without any modification. The scalable nature of our methods allows infrastructure administrators to add as many counters of service-sensitive events as they wish to. Everything else in the monitoring process will be taken care of automatically with no need for further tuning.

In a larger context, the open question addressed in this paper is whether large infrastructures should be prepared to recover from "unavoidable failures," as is commonly suggested. Even when advanced recovery mechanisms exist, they are often not tested due to the risk involved in testing live environments. Indeed, advanced recovery (beyond basic failover) testing of large-scale systems is extremely complicated and failure prone, and rarely covers all faulty scenarios. Consequently,

some outages of Amazon EC2 [1], Google's search engine, and Facebook, and even the Northeast power blackout of 2003, were attributed to the cascading recovery processes, which were interfering with each other during the handling of a local event. It is conceivable that there exist large systems whose recovery processes have never been tested properly.

Like [3], we propose an alternative: proactively treating latent faults could substantially reduce the need for recovery processes. We therefore view this work as a step towards more sensitive monitoring machinery, which will lead to reliable large-scale services.

ACKNOWLEDGMENTS

This work was conducted while Moshe Gabel and Assaf Schuster were visiting Microsoft Research. The authors wish to thank the EU LIFT project, supported by the EU FP7 program.

REFERENCES

[1] N. Palatin, A. Leizarowitz, A. Schuster, and R. Wolff, "Mining for misconfigured machines in grid systems," in *Proc. SIGKDD*, 2006.
[2] 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.
[3] A. B. Nagarajan and F. Mueller, "Proactive fault tolerance for HPC with xen virtualization," in *Proc. ICS*, 2007.
[4] P. Bodík, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: Automated classification of performance crises," in *Proc. EuroSys*, 2010.
[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] M. Isard, "Autopilot: automatic data center management," *SIGOPS Oper. Syst. Rev.*, 2007.
[7] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam, "Critical event prediction for proactive management in large-scale computer clusters," in *Proc. SIGKDD*, 2003.
[8] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *Proc. ICAC*, 2004.
[9] 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.
[10] A. J. Oliner, A. Aiken, and J. Stearley, "Alert detection in system logs," in *Proc. ICDM*, 2008.
[11] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *Proc. USENIXATC*, 2010.
[12] 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.
[13] M. P. Kasick, J. Tan, R. Gandhi, and P. Narasimhan, "Black-box problem diagnosis in parallel file systems," in *Proc. FAST*, 2010.
[14] C. McDiarmid, "On the method of bounded differences," *Surveys in Combinatorics*, 1989.
[15] W. J. Dixon and A. M. Mood, "The statistical sign test," *Journal of the American Statistical Association*, 1946.
[16] R. H. Randles, "A distribution-free multivariate sign test based on interdirections," *Journal of the American Statistical Association*, 1989.
[17] J. Tukey, "Mathematics and picturing data," in *Proc. ICM*, 1975.
[18] T. M. Chan, "An optimal randomized algorithm for maximum Tukey depth," in *Proc. SODA*, 2004.
[19] J. A. Cuesta-Albertos and A. Nieto-Reyes, "The random Tukey depth," *Journal of Computational Statistics & Data Analysis*, 2008.
[20] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," *SIGMOD Rec.*, 2000.
[21] C. D. Manning, P. Raghavan, and H. Schüze, *An Introduction to Information Retrieval*. Cambridge University Press, 2008.
[22] 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.
[23] 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.
[24] 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.
[25] C. Huang, I. Cohen, J. Symons, and T. Abdelzaher, "Achieving scalable automated diagnosis of distributed systems performance problems," HP Labs, Tech. Rep., 2007.

[1] http://aws.amazon.com/message/65648/