Latent Fault Detection in Cloud Services

Mickey Gabel[a,b,1], Ran Gilad-Bachrach[a], Nikolaj Bjørner[a], Assaf Schuster[a,b,1]

[a] *Microsoft Research, Redmond, USA*
[b] *Computer Science Department, Technion, Haifa, Israel*

1. Introduction

Large scale internet cloud services comprising of thousands of computers are ubiquitous. With so many machines, it is not reasonable to assume that all of them are working properly and are well configured [13]. If faults are left unnoticed they might accumulate to the point where redundancy and fail-over mechanisms break. Therefore, detecting *latent faults* is essential for preventing failures and increasing the reliability of cloud services. 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.

Monitoring for faults is usually done by collecting and analyzing performance counters [1, 4, 10]. 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). Manual monitoring of large datacenters is impractical due to the large number of machines to track and the large number of counters to follow. Therefore, some level of automation is necessary.

Existing automated techniques for detecting failures are mostly rule-based. A set of watchdogs [10] is defined. A watchdog typically monitors a single counter on a single machine or service. For example, a watchdog may monitor the temperature of the CPU or the amount of free disk space. Whenever a predefined threshold is crossed a certain action is trigged. These actions range from notifying the system operator to automatic recovery attempts.

Rule-based failure detection suffers from several key problems. For example, thresholds need to be made low enough such that faults will not be left unnoticed. At the same time, they should not be set too low 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, these are done only as a consequence of "postmortem" examination.

Others have noticed the shortcomings of these rule-based approaches. [5, 6] proposed training a detector on historic annotated data. However, these approaches fall short due to the difficulty in obtaining this data, as well as the lack of flexibility 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 [13]. This approach is intrusive and hence prohibitive in many cases.

Thus, the challenge for designing latent fault detection mechanism is to design it in a way that will be agile enough to handle the variations in the service and the differences between services. It should also be non-intrusive yet efficient in the sense that it will detect as many faults as possible with only few false alarms. As far as we know, we are the first to propose a framework and methods that addresses all these issues simultaneously.

Contribution

We develop a domain independent framework for identifying *latent faults*. A machine is considered to have a latent fault if it is defective but not yet failing. Not all machine failures are the result of latent faults. Power outages, network outages and malicious attacks can occur instantaneously with no incubation period. We make no attempt at predicting these failures.

Our framework is unsupervised and non-intrusive. The main idea behind it is to compare machines performing the same task at the same time. A machine that deviates from the common behavior is flagged as suspicious. This technique is agile; we demonstrate its ability to work on different services without need for any tuning. Moreover, changes in the workload or even changes to the service itself do not affect its performance. We demonstrate three tests within this framework and provide strong theoretical guarantees on the false detection rates of the proposed tests. We also evaluate them on several production services of various sizes and natures, including large scale services, as well as a service that is using virtual machines.

[1] This work was conducted while the author was visiting Microsoft Research

2. Related Work

The problem of machine failure detection was studied by several researchers in recent years, and proposed techniques have so far been mostly supervised.

Chen et al. [4] separate counter values into workload counters and internal measurements (system output), and analyze the correlation between these sets of measurements. They introduce *principle canonical correlation analysis*, and use it to transform the measurements into two maximally correlated subspaces with high variance. These correlations are then tracked over time. This approach requires training the system to model baseline correlations, and requires domain knowledge when choosing counters

Chen et al. [5] presented an approach based on learning decision trees. They successfully applied it to a large real-world service. However their system is supervised, requiring labeled examples of failures and domain knowledge. Labeled examples are hard to obtain. Moreover, supervised approaches are less adaptive to workload variations and changes to the software and hardware.

Pelleg et al. [14] explore failure detection in virtual machines. They describe a supervised system based on decision trees that monitors a set of 6 carefully selected hypervisor counters, instead of monitoring the guest OS and application counters directly. Though the basis is domain independent, it is supervised and requires training on labeled examples. Moreover, counters are manually selected from hypervisor counters. It is therefore not immediately portable, and only suitable for well managed settings that include predictable workloads and previously-seen failures.

Bodík et al. [1] provide a technique to identify and classify system performance crises – points in time where the system performance falls below accepted values. They produce fingerprints from aggregate counters that describe the state of the entire datacenter, and use these fingerprints to identify crises. Their technique also provides direct control over the false positive rate. As with other supervised techniques, the approach in [1] requires labeled examples. In addition, Bodík et al. concentrate on quick detection of existing failures, while we focus on detection of latent faults ahead of machine failures. Finally, they study service level faults, while we focus on machine level faults.

Cohen et al. [6] similarly use machine learning approach to induce a tree-augmented Bayesian network classifier. This approach requires a training set with labels but no other domain knowledge, and is effective without service-specific counters. They also identify the relevant counters to give insight into the problem with the service. However the classifier is very sensitive to changing workloads, due to its reliance on a training set. As with [4], it was designed for monitoring a system and is not applicable to identifying failing machines in datacenters. Likewise, it was evaluated on an experimental setup with synthetic workloads, and not a real-world system. Ensembles of models are used in [18] to reduce the sensitivity of the former approach to workload changes, at the cost of decreased accuracy when there are too many types of problems ([9]).

Finally, Palatin et al. [13] employ a data mining approach and introduce GMS (Grid Monitoring System). GMS uses a distributed version of the HilOut algorithm, which detects outliers using the average distance to k nearest neighbors. Similar to our method, GMS is based on outlier detection and is unsupervised and requires no domain knowledge. However GMS is intrusive - it requires sending jobs to be run on the monitored hosts, essentially modifying the running service. Moreover it provides no statistical guarantees on false detection rates.

3. Framework and Methods

The scalability and reliability of cloud services is achieved in many cases by means of duplication. That is, the service is duplicated on multiple machines and there exists some load balancing process that splits the workload between these machines. Therefore, we expect all machines that perform the same role, using similar hardware, to exhibit similar behavior as it is being reflected by performance counters. Whenever we see a machine that consistently differ from the rest of the machines, we flag it as a suspect for having a latent fault.

To compare machines, we use performance counters. Machines in datacenters often periodically report and log a wide range of performance counters. These counters are collected both from the hardware (e.g., temperature), the operating system (e.g., number of threads) and from the application. However, since we do not assume domain knowledge, we treat all counters equally.

We model the problem in the following way: there are M machines and each reports 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 process that generated this sample for machine m is the same process that generate the sample for any other machine m'. However, if we see that the sample for machine m is significantly different than the sample for other machines, we reject the hypothesis and flag the machine as suspicious.

In the context of the framework we propose three tests to detect latent faults. We compare them in Section 4. The tests are made of two stages. The preprocessing stage (see Alg. 1) is common to all the tests. In this stage the data is collected, and the vectors $x(m,t)$ are formed. In

the second stage the data is analyzed and a p-value is computed for each of the machines. 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 for a given machine falls below a predefined significance level α, the null-hypothesis can be rejected, and the test reports this machine as a suspect. We propose three possible ways to implement the second stage: the *sign test* (Alg. 2), the *Tukey test* (Alg. 3) and the *LOF test* (Alg. 4).

The rest of this section is devoted to describing the rationale behind these algorithms and proving their correctness.

3.1. Notation

The cardinality of a set S is written $|S|$, while for a scalar s, we use $|s|$ as the absolute value of s. The L_2 norm of a vector x is $\|x\|$ and $x \cdot x'$ is the inner product of x and x'.

We use $\mathcal{M}$ as the set of all machines in a test, m, m' refer to specific machines, and $M = |\mathcal{M}|$ is the number of machines. In the same fashion, $\mathcal{C}$ is the set of all counters, c refers to a specific counter, and $C = |\mathcal{C}|$. We slightly abuse notation where in Alg. 1, $\mathcal{C}$ refers to all the counters available in the system while in the rest of the discussion, $\mathcal{C}$ refers only to the counters that were selected by Alg. 1. Finally, $\mathcal{T}$ are the time points where counters are sampled, t, t' refer to specific time points, and $T = |\mathcal{T}|$. In our experiments time points are sampled every 5 minutes during a 24 hours interval.

3.2. Framework Assumptions

While modeling the problem we make several assumptions that we will now make explicit.

- The majority of the machines are working properly at any given point in time.
- Machines are homogeneous in the sense that they perform a similar task and use similar hardware, software, and operating system.
- Workload is balanced across all machines.
- All counters are ordinal and are reported in the same rate.
- Counter values are memoryless in the sense that they depend only on the current time period (and independent of the identity of the machine).

Formally, we assume the $x(m, t)$ is a realization of a random variable $X(t)$ whenever the machine m is working properly. The idea is that since all machines perform the same task, and since the load balancer tries to split the load evenly between the machines, we should expect them to perform similarly. We do expect to see changes over time due to changes of the workload for example. However, we expect these changes to be reflected in all machines in a similar way. Therefore, whenever we see a machine which violates this assumption we flag it as a suspect.

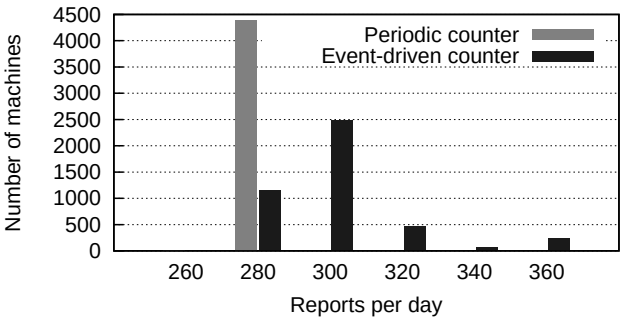


Figure 1: Histogram of number of reports for two kinds of counters. The report rate of the event-driven counter has a higher variance.

3.3. Preprocessing

From the outset it is clear that our model is simplified, and in practice not all of its assumptions about counters hold. We observe three key violations of the assumptions we make. First, not all counters are reported on a fixed period. Second, each periodic counters might have a different period. And third, 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. In order to eliminate these problems as much as possible, the preprocessing algorithm (Alg. 1) creates a representation of the data which better reflects the assumptions.

The first issue addressed is non-periodic counters, such as event-driven counters. These might create biases from the point of view of our statistical tests, so we detect and remove these counters. Such counters will typically have a different number of reports on different machines as is demonstrated in Figure 1. The detection is done in the following way: for a counter c, we measure the number of times it is reported on every machine m and denote this number by $n_c(m)$. We expect all machines to have similar number of reports for a periodic counter. However, since some machines might be faulty, we need a robust way to detect that. Therefore, we compute the median of the number of reports and denote it by n_c. To compute the amount of variability in the number of reports, we compute the 90^{th} percentile of $|{}^{(n_c(m)-n_c)}/_{n_c}|$ and eliminate counters for which this number is too large. In our implementation this threshold is set to 0.01. The choice of the

number 90 represents our assumption that at least 90% of the machines are working properly. We also eliminate infrequent counters. In our implementation, counters that are being reported less than 6 times a day are ignored.

Even periodic counters are not necessarily being reported at the exact same rate and exact same time across all machines. To address that, we select a set of time intervals, and record for each machine and counter the last reported value before the end of each interval. This creates an alignment of the reports across machines and counters. In our implementation we use equal time intervals of 5 minutes.

Next, we normalize each counter to have a zero mean and a unit variance. This is done to eliminate artifacts of scaling and numerical instabilities.

The last issue addressed is counters that violate the memoryless assumption. We expect that a counter will have similar mean when measured on different machines (see, e.g. Figure 2). A counter that violates this assumption is ignored. However, if we eliminate all counters on which there are some deviations of the mean, we will not be able to detect the deviations that are due to malfunctioning machines. Therefore, we use robust statistics in the following way; we compute the median $\mu_c(m)$ of the value of the counter c on the machine m. We compute the median μ_c of the median values. To provide the right scaling for the distance, we use a robust version of standard deviations: the *Median Absolute Deviation* (MAD) (see, e.g., [16]). The MAD of a sample S is defined as

$$\mathrm{MAD}(S) = \mathrm{median}_{s\in S}\left(|s - \mathrm{median}_{s\in S}(s)|\right).$$

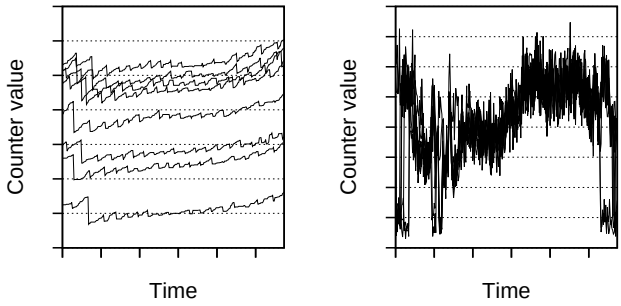
Finally, compute the 90^{th} percentile of

$$\left|\frac{\mu_c(m) - \mu_c}{\mathrm{mad}_c(m)}\right|$$

where $\mathrm{mad}_c(m) = \mathrm{MAD}_{t\in\mathcal{T}}(\{x_c(m,t)\})$. We ignore counters for which this value is greater than a threshold (2 in our experiments).

Although we eliminate counters during preprocessing, we are left with many useful counters, typically more than one hundred (see Table 4). Moreover, this process is of great importance when monitoring virtual machines. It allows eliminating counters that reflect cross-talk between virtual machines running in the same physical machines, and focus on the counters which are representatives of the actual performance of the virtual machine of interest.

The preprocessing algorithm has few parameters that need to be tuned. However, as demonstrated in Section 4.6, the algorithm is robust and moderate changes to their values do not affect its performance in any significant way.



(a) Large mean diversity (b) Small mean diversity

Figure 2: Counter values for 8 machines. The counter in 2(a) shows different biases for individual machines. Despite the variance over time, in 2(b) all machines act in tandem.

Algorithm 1: The preprocessing algorithm. Receives the raw counters, eliminates problematic counters and normalizes the data. We denote by $p_{90}(S)$ the 90^{th}-percentile of S.

Let $\theta^1 = 0.01, \theta^2 = 2, \theta^n = 5$;
Let $z_c(m,t)$ = the last value of counter c on machine m before time t;
Let $n_c(m)$ = number of reports for c on m;
foreach *counter* $c \in \mathcal{C}$ **do**
  $\nu_c \leftarrow \mathrm{mean}_{m\in\mathcal{M},t\in\mathcal{T}}(z_c(m,t))$;
  $\sigma_c \leftarrow \mathrm{STD}_{m\in\mathcal{M},t\in\mathcal{T}}(z_c(m,t))$;
  foreach *machine* $m \in \mathcal{M}$ **and** *time* $t \in \mathcal{T}$ **do**
    $y_c(m,t) \leftarrow \frac{z_c(m,t)-\nu_c}{\sigma_c}$;
  end
  foreach *machine* $m \in \mathcal{M}$ **do**
    $\mu_c(m) \leftarrow \mathrm{median}_{t\in\mathcal{T}}(y_c(m,t))$;
    $\mathrm{mad}_c(m) \leftarrow \mathrm{MAD}_{t\in\mathcal{T}}(y_c(m,t))$;
  end
  $n_c \leftarrow \mathrm{median}_{m\in\mathcal{M}}(n_c(m))$;
  $\psi_c^1 \leftarrow p_{90}\left(\left|\frac{n_c(m)-n_c}{n_c}\right|\right)$;
  $\mu_c \leftarrow \mathrm{median}_{m\in\mathcal{M}}(\mu_c(m))$;
  $\psi_c^2 \leftarrow p_{90}\left(\left|\frac{\mu_c(m)-\mu_c}{\mathrm{mad}_c(m)}\right|\right)$;
  if $(\psi_c^1 \le \theta^1)$ **and** $(n_c \ge \theta^n)$ **and** $(\psi_c^2 \le \theta^2)$ **then**
    foreach *machine* $m \in \mathcal{M}$ **and** *time* $t \in \mathcal{T}$ **do**
      $x_c(m,t) \leftarrow y_c(m,t)$;
    end
  end
end

3.4. Statistical Tests

After preprocessing, we use statistical tests to detect the outliers. We present three such tests and compare them in Section 4.

3.4.1. The Sign Test

The sign test [8] is a classical statistical test. It verifies the hypothesis that two samples share a common median. It has been extended to the multivariate case [15]. We extend it further to allow the comparison of multiple machines simultaneously.

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. Assuming that 0 divided by 0 is 0, the vector

$$\frac{x(m,t) - x\left(m',t\right)}{\|x(m,t) - x\left(m',t\right)\|}$$

has length at most 1. If the two machines, m and m' are indeed working properly, the expected value of this random variable is zero. The sum of several samples over time is therefore also expected not to grow far from zero.

Algorithm 2: The sign test. Given a significance level α, outputs a list of suspicious machines.

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

In our setting, we have a vector $x(m,t)$ for every machine $m \in \mathcal{M}$ and every time $t \in \mathcal{T}$. To test machine m, we compute the vector

$$v(m) = \frac{1}{T(M-1)} \sum_{t\in\mathcal{T}} \sum_{m'\in\mathcal{M}} \frac{x\left(m,t\right) - x\left(m',t\right)}{\|x\left(m,t\right) - x\left(m',t\right)\|} \quad .$$

The following theorem shows that if all machines are working properly, the norm of $v(m)$ is expected to be close to its empirical mean on all machines.

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* $v(m)$ *be as defined above. Then for every* $\gamma > 0$

$$\Pr\left[\exists m \in \mathcal{M} \text{ s.t. } \|v(m)\| \geq \hat{E}\left[\|v(m)\|\right] + \gamma\right] \leq$$

$$(M+1)\exp\left(-\frac{TM\gamma^2}{2\left(\sqrt{M}+2\right)^2}\right) \quad ,$$

where $\hat{E}\left[\|v(m)\|\right]$ *is the empirical mean of* $\|v(m)\|$*.*

Theorem 1 provides p-values for the sign test. It shows that if all machines are working properly $\|v(m)\|$ does not deviate much between machines. Hence, Theorem 1 provides a guarantee that the amount of false detections should be small. The main tool used in the proof of the theorem is the bounded sequence inequality [12]. The complete proof is omitted due to lack of space.

A beneficial property of the sign test is that whenever it detects a suspect machine, it also provides a fingerprint for the failure. The vector $v(m)$ scores every counter. The test provides 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 is lower than the rest of the population. This fingerprint can be used as a way to identifying recurring types of failures [1]. It can also be used as a starting point for root cause analysis. The entries in $v(m)$ with the highest absolute values point to the counters with large deviations from normal behavior. Hence, they can serve as a starting point for investigation.

3.5. The Tukey Test

The Tukey test is based on a different statistical tool, the Tukey depth function [17]. Given a sample of points, the Tukey depth function gives high scores to points which are at center positions in the sample and low scores to points which are in the perimeter. For a point s, we go over all possible half-spaces that contains the point s and count the number of points of S in the half-space. The depth is defined as the minimum number of points over all possible half-spaces; formally let X be a set of points in the vector space R^d and $x \in R^d$, then the Tukey depth of x in X is:

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

In our setting, we say that if the vectors $x(m,t)$ for a fixed machine m consistently have low depths, then m is likely to be behaving differently than the rest of the machines. However, there are two main obstacles. First note that each point in time, the size of the sample is exactly the number of machines M and the dimension is

Algorithm 3: The Tukey test. Given a significance level α it outputs a list of suspicious machines.

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

the number of available counters C. In many cases the dimension C is larger than the number of points M and therefore, it is 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 [3]. Therefore, similar to [7], we select few projections of the data to low dimensions and compute depths in the low dimension. In our case we project the data to R^2 and compute the Tukey depth of the projected points.

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 time t we project $x(m,t)$ for all the machines $m \in \mathcal{M}$ and compute the depth in the projected space. Note that in R^2, computing the depth of all M points has a complexity of only $O\left(M \log(M)\right)$. We repeat the process several times, with different projections to obtain the depth $d(i,m,t)$ for machine m at time t with the i'th projection. Finally, we compute the score for machine m to be

$$s(m) \leftarrow \frac{2}{TI\left(M-1\right)} \sum_{i=1}^{I} \sum_{t \in \mathcal{T}} d\left(i,m,t\right) \quad . \tag{1}$$

If all machines behave appropriately, $s(m)$ should be concentrated around its mean. However, if a machine m, has a much lower score than the empirical mean, this machine is flagged. The following theorem shows the correctness of this approach.

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* $s(m)$ *be as defined in (1). Then for every* $\gamma > 0$

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

where $\hat{E}\left[s(m)\right]$ *is the empirical mean of* $s(m)$.

Theorem 2 provides the way to compute p-values for the Tukey test. Similar to the proof of Theorem 1, it uses the bounded sequence inequality [12]. Proof is omitted due to lack of space.

3.6. The LOF Test

The LOF test is based on the *Local Outlier Factor* (LOF) algorithm [2] which is a popular outlier detection algorithm. The LOF function tries to find outliers by only looking at local neighborhoods. The assumption is that on different areas, the density of the sample might be different and this does not imply that points in less dense areas are outliers. The score of the LOF function is not calibrated. The greater the LOF score is, the more suspicious the point is. However, the precise value of the score has no particular meaning.

Algorithm 4: The LOF test. Given a significance level α it outputs a list of suspicious machines. This test uses the LOF algorithm [2] to score the machines.

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

To avoid falsely flagging machines, we integrate the LOF score over time in the following way: At every point in time we compute $l(m,t)$, the LOF score of machine m at time t. Next, we record the rank $d(m,t)$ of the LOF score of machine m at time t. The rank $r(m,t)$ is such that the machine which had the lowest LOF score will have rank 0, the second lowest will have the rank 1 and so on. If all machines are working properly, the rank $d(m,t)$ should be distributed uniformly on $0, 1, \ldots, M-1$. Therefore, the statistic

$$s(m) = \frac{2}{T(M-1)} \sum_{t \in \mathcal{T}} d(m,t) \tag{2}$$

has an expected value of 1. If the score is much higher, the machine is flagged. The correctness of this appraoch 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 $s(m)$ be as defined in (2). Then for every $\gamma > 0$*

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

Proof. Define $X_t = \{x(m,t)\}_{m \in \mathcal{M}}$. Consider a fixed machine m, we can view $s(m)$ as a random variable over the X_t's. Note that if $\{X_t\}$ and $\{X'_t\}$ differ only on X_{t^*} then the value of $s(m)$ changes by at most $2/T$. Using the bounded sequence inequality [12] we obtain the stated result. □

4. Empirical Evaluation

We conducted several experiments to validate our assumptions about latent faults and to compare the different methods. The main experiments were made on production services of various sizes.

4.1. Production Services

The production services used in the experiments are live, real-world, distributed services. These services have different characteristics. The LG ("large") service consists of a large cluster ($\sim$ 4500 machines) which is a part of the index service of a leading search engine. The PR ("primary") service runs on a mid-sized cluster ($\sim$ 300 machines) and provides information about previous interactions of users. Technically, 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.

These services run on top of a data center management infrastructure which provides services such as deployment of services, monitoring and automatic repair. We use the automatic repair log to deduce information concerning the machines' health signals (clearly, this information is incomplete in many ways). This infrastructure also collects different performance counters both from the hardware and the running software. We use these counters as the source of information for our experiments.

4.2. Protocol Used for Experiments

We applied our methods to each service independently and in a daily cycle. That is, a day's worth of data was collected and used to flag suspect machines using each of the tests. To avoid overfitting, parameters where tuned based on historical data of the SE service. The significance level α was fixed at 0.01 and the threshold value for the feature selection mechanism were fixed.

To evaluate the performance of our methods we compared latent faults to machine health signals as reported by the infrastructure at a later day. Health events are raised based on rules for detecting software and hardware failures. Our thesis is that some latent faults will evolve over time to 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 use existing health systems to verify the results. Where they are incomplete we used manual inspection of counters and audit logs

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 method are referred to as *suspects*. Borrowing from the information retrieval literature [11], we use *precision* to measure the fraction of failing machines out of all suspect machines and *recall* (also called *True Positive Rate*, or TPR) to measure the fraction of suspect machines out of all failing machines. We also use the *False Positive Rate* (FPR) to denote the fraction of healthy machines out of all suspect machines. Formally speaking, these are conditional probabilities where recall = Pr(suspect | failing), FPR = Pr(suspect | healthy) and precision = Pr(failing | suspect). A summary of the terms used is provided in Table 1.

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

Table 1: Summary of terms used in our experiment reports.

4.3. Results

We discuss the results for each services separately due to their different nature. We do that despite the fact that we applied the same techniques to each service with the same choice of parameters.

4.3.1. The LG Service

Table 2 shows a summary of the results of the experiment on the LG service. This table shows how well the different methods predicted machine failures in the service. The low false positive rate (FPR) reflects our design choice to minimize false positives. Tracking the precision results proves our assumption that latent faults exist 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 that are flagged as suspects by this method will indeed fail during the next two weeks. Moreover, the majority of these failures happen in the second day or later, showing that the prediction horizon is longer than the immediate day.

The recall numbers in Table 2 indicate that approximately 20% of the failures in the service where already manifested in the environment for circa a week before they were detected (see a more comprehensive study in Section 4.4).

The cumulative failure graph (Figure 3) depicts the fraction of suspect machines across all days that failed up to n days after the detection of the latent fault. In other words, it shows precision vs. prediction horizon. The "total" column 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 suspects at random based on the failure probability in the LG service. These graphs demonstrate again the existence of latent faults in the environment.

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. These curves are generated by varying the significance level (α). The resulting curves are shown in Figure 4.

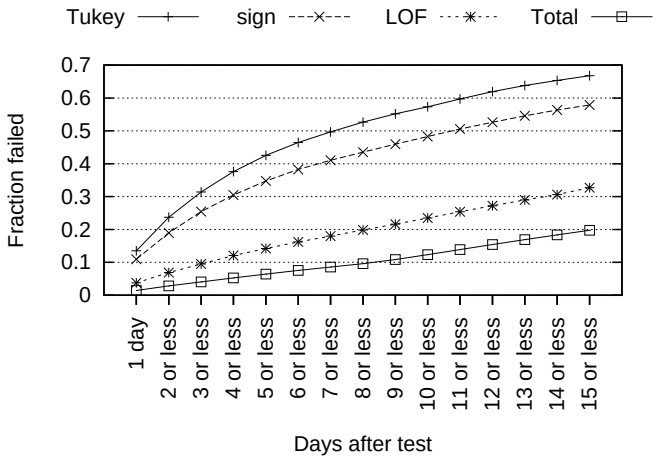


Figure 3: Cumulative failures on LG service. Most of the faults detected by the sign test and the Tukey test become failures only several days after they have been detected.

Both 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). [2]

The dips in the beginning of the P-R curves reflect machines that consistently get low p-values, but do not fail. We have manually investigated some of these machines, and they can be divided into (1) machines that have failed, but the service did not notify the platform about it (incomplete logs), and (2) machines that are misconfigured or underperforming, but are not outright failing since the LG service does not monitor for these conditions. In a

[2] One might expect that with a significance level of 0.01 the false positive rate will be less than 1%. However, we have manually confirmed cases of actual latent faults that go unnoticed in the environment. Due to the verification protocol, such machines are consistently considered false positives but are actually correctly flagged as faulty machines.

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

Table 2: Prediction performance on LG with significance level of 0.01

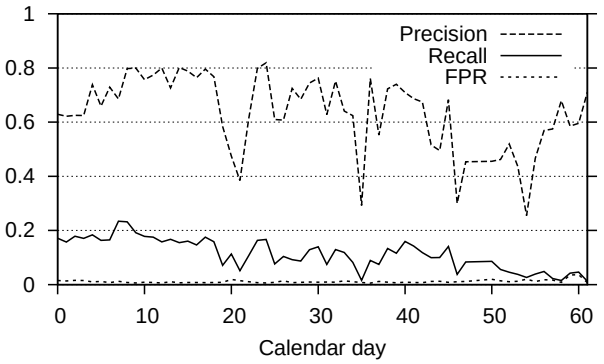


Figure 5: Tukey performance on LG across 60 days, with 14-day horizon. It shows the test is not affected by changes in the workload. Test quickly recovers from service updates on days 22 and 35. Lower performance on days 45–55 is an artifact of gaps in counter logs and updates on later days.

sense these are "soft" failures that may not result in machine failure.

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 14-day horizon for each calendar day (Figure 5). 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 upgrades that were made to the service during these times. Since the upgrade is not done simultaneously on all machines, the test detects the divergence between the versions and reports these as failures. However, once the upgrade completed, no tuning was necessary for the test to regain its performance.

4.3.2. PR and SE Services

The SE service only mirrors data written to PE, but serves no read requests. Since SE machines serve only write requests, they are not as loaded as PR machines which serve both read and write requests. Hence, traditional rule-based monitoring system are less likely to detect failures on these machines. Therefore, the existence of latent faults on these machines is likely to be manifested only when there is a failure in a primary machine and the faulty SE machine is converted to the primary (PR) role.

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

Table 3: Prediction performance on SE with 14-day horizon at significance level 0.01

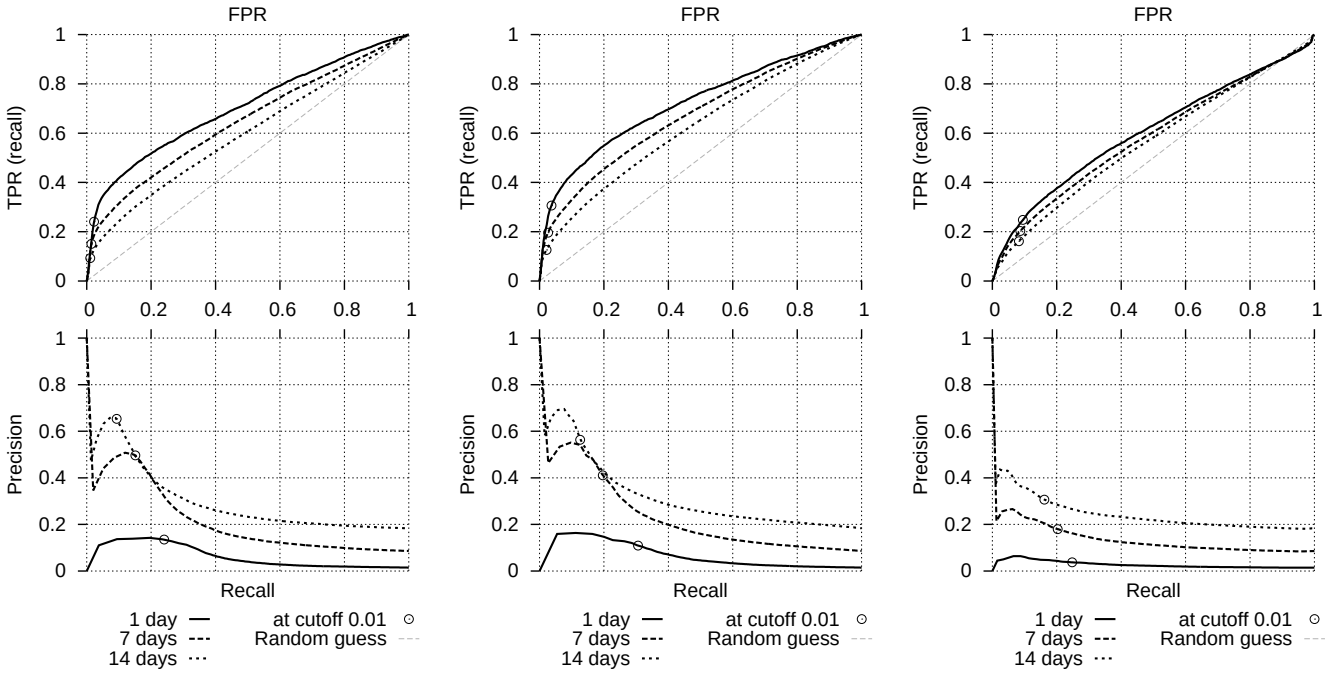
Unfortunately, the "health monitors" for the PR and SE services are not as comprehensive as the ones for the LG service. Since we use the "health monitors" as the objective signal against which we measure the performance of our tests, these measurements are less reliable. To compensate for that, in some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

Performance on SE service for significance level of 0.01 are summarized in Table 3. ROC and P-R curves are in Figure 6. It is clear that our methods are able to detect and predict machine failures, therefore we conclude that latent faults do exist in this service as well, even if to a lesser extent. Since this is a backup service, we believe 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 PE service is similar to the SE but even more acute. The amount 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.

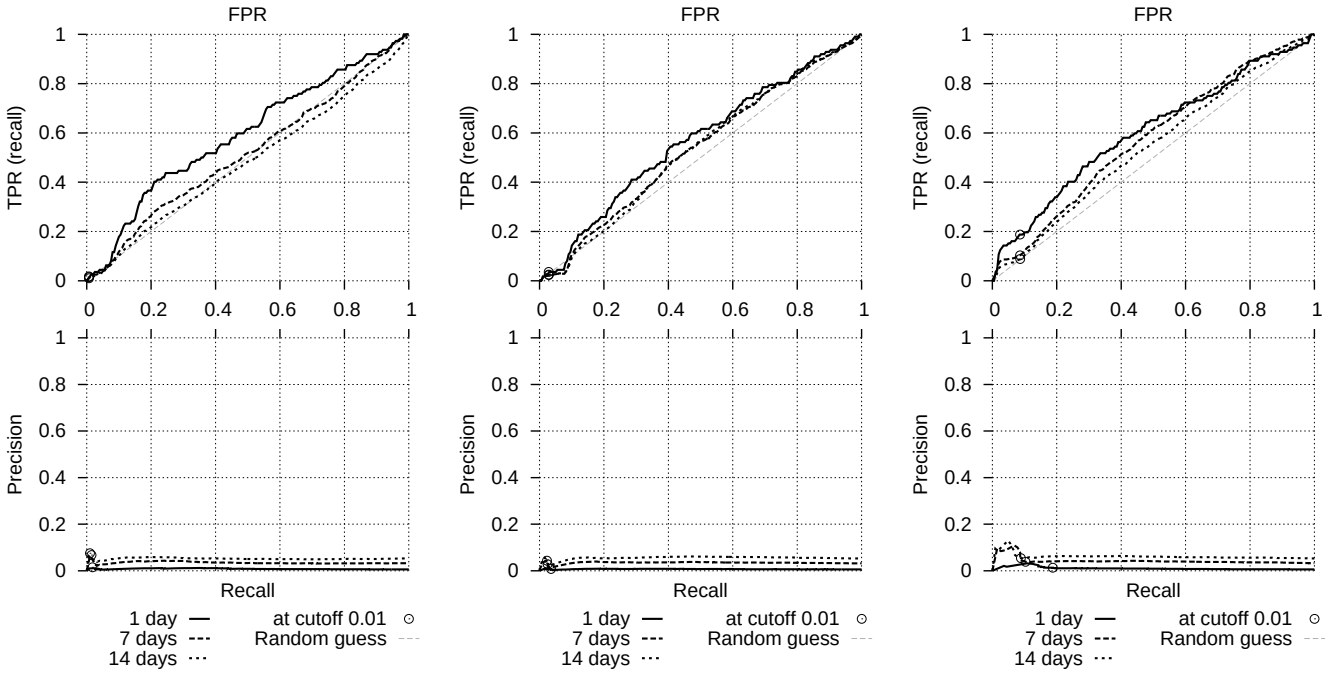
4.3.3. VM Service

The VM service presents a greater challenge, due to the use of virtual machines and small population of machines. The risk is that a test will flag machines as suspects 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



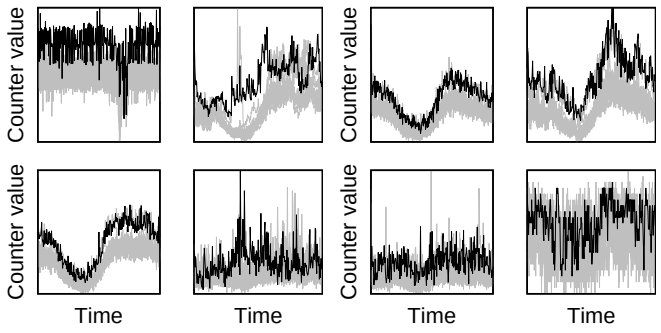
(a) Tukey (b) sign (c) LOF

Figure 4: ROC and P-R curves on LG service



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

Figure 6: ROC and P-R curves on the SE service



(a) Detection day (b) Warning day

Figure 7: Aberrant counters for suspect VM machine (black) compared to other 14 machines (gray).

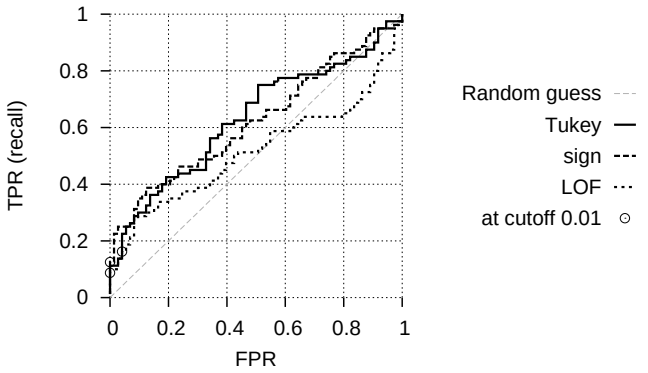


Figure 8: Detection performance on LG service. At least 25% of failures have preceding latent faults.

CPU usage, thread count, disk queue length and other counters indicating a large workload. Indeed, two days after detection there was a watchdog warning indicating that the machine is overloaded. The relevant counters for this machine are plotted in Figure 7. The second machine with detected latent fault 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.

4.4. Estimating the Amount of Latent Faults

We conducted a controlled experiment to estimate the amount of latent faults in the LG service. As discussed earlier, 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. Hence, the goal of this experiment is to estimate the percentage of failures which do have a latent period in the current environment.

We selected 80 failure events at random and checked if 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 which are known to by healthy. For both sets we require that events come from different machines, and from a range of times and dates.

For the sake of 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 days period of our investigation.

Figure 8 shows the ROC curves for this experiment. Failing machines where latent faults are detected are true positives. Healthy machines that are flagged as suspects are counted as false positives. Both the sign test and the Tukey test 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 have a significant period of time in which they are latent in the environment.

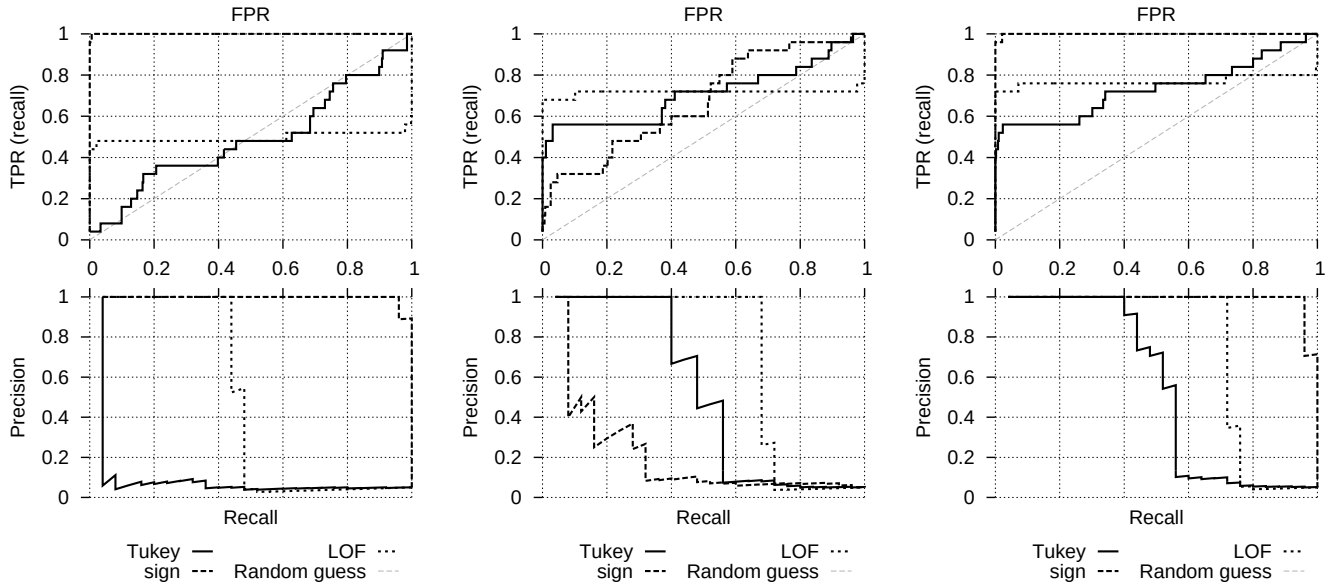
4.5. Comparison of Tests

Figure 4 shows the ROC and P-R curves for the LG service experiment. The outliers detected by LOF are less correlated with real failures when compared to Tukey and sign. The curves for Tukey and sign are very similar. However, for the specific significance level of 0.01 Tukey is slightly better as seen, for example, in Figure 3. On the SE service, Tukey performs slightly better than the other methods (see Figure 6), while on the VM service, Tukey and sign are very similar in performance while LOF is the least performing of the three.

To measure agreement between the different tests we compared the scores they assign to the same machine. Indeed, the tests tend to assign low p-values for similar sets of suspect machines, despite being based on different principles.

To better characterize the sensitivities of the different tests we evaluated them on artificially generated data. We generated random data equivalent to 500 machines, each reporting 150 counters every five minutes for a full day. Counters are based on a base offset to which we add random normal noise of different scale (per counter) but no offset. The base offset varies across the day to simulate the daily workload changes that machines go through, and each counter has a different workload scale and and offset. Figure 9 shows several such counters.

We define three types of "faults": location (offset), scale, or both (location+scale). 25 machines were selected as failing machines, and for these machines 10% of counters are noisy with either different offset, scale or both. The strength of the difference varies across the failing machines, and we compare sensitivity of each test to different



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

Figure 10: Performance on types of synthetic fault.

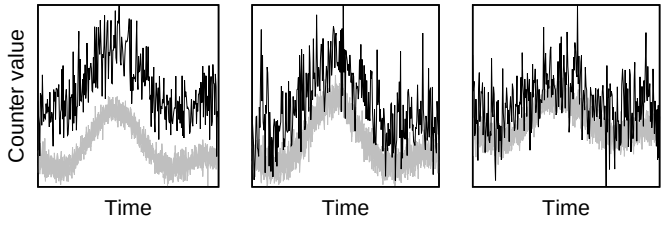


Figure 9: Several synthetic counters for 8 machines. The highlighted machine has a "fault".

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

Table 4: Average number of counters. The first rows present counters that were eliminated in the preprocessing. Despite the automated filtering, many counters remain for detections.

kinds of faults by measuring its performance. The curves are shown in Figure 10.

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 changes. When scaling is changed, LOF is more sensitive at the regime of low false positive rates but does not do well later on. Tukey is more sensitive than sign to scaling changes.

4.6. Filtering Counters

In the first stage of processing, we filter out counters based on their statistics. Table 4 reports the average number of counters used in each experiment.

When filtering counters, we only keep counters for which the mean diversity ψ_c^2 is equal or less than the threshold 2. Because ψ_c^2 is scaled using median absolute deviation, it is reasonable to select small values as the threshold. The threshold value 2 was chosen based on our conservative design choice to miss failures while rejecting counters with diverse means (since they can cause false positives).

Figure 11 justifies this choice, as the majority of counters that were not filtered on earlier stages have $\psi_c^2 \leq 2$ on most services, and the range of 2–10 contains few counters.

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 diversity thresholds in the range 2–10. With strict significance level, higher thresholds result in slightly better recall but slightly lower precision, confirming our expectations.

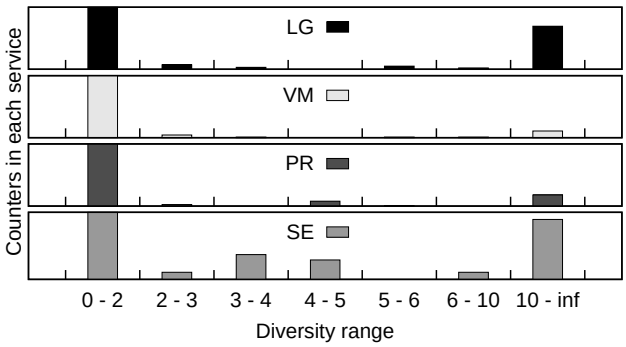


Figure 11: Histogram of counter mean diversity for all services. Majority of counters are below 2.

5. Conclusions

In this work we introduce a novel framework for detecting latent faults and derive three tests that act in this framework. The significance of this framework is that it allows us to address the problem of latent fault detection in a way that is agile enough to be used across different systems and to withstand changes over time. As far as we know, we are the first to address this issue.

We proposed tests to detect latent faults and proved efficient guarantees on their false detection rates. These methods were evaluated on several production services of different nature. Our methods were able to detect many latent faults days ahead of rule-based watchdogs. Moreover, we have shown that our tests are 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. Moreover, our experiments validate that latent faults are common even in well-managed datacenters.

Comparing the three proposed tests, we observed that the sign test and the Tukey test were more accurate than the LOF test. The differences between the sign test and the Tukey test were not significant. Additionally, the sign test can uniquely generate fingerprints for detected faults. These fingerprints allow better understanding of the root cause. They can be used both to group together similar faults and to identify the counters that deviate the most.

There are many directions for future study; we would like to be able to monitor non-homogeneous datacenters, where machines are different or perform different tasks. New tests can be introduced that will handle event-driven counters and counters with varying report rates with a greater rigor. Another interesting direction is to study which types of faults tend to have long incubation time as opposed to more sudden failures.

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

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. *SIGMOD Rec.*, 2000.

[3] T. M. Chan. An optimal randomized algorithm for maximum Tukey depth. In *Proceedings of SODA*, 2004.

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

[5] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. *Autonomic Computing, International Conference on*, 2004.

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

[7] J. A. Cuesta-Albertos and A. Nieto-Reyes. The random Tukey depth. *Journal of Computational Statistics & Data Analysis*, 2008.

[8] W. J. Dixon and A. M. Mood. The statistical sign test. *Journal of the American Statistical Association*, 1946.

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

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

[11] C. D. Manning, P. Raghavan, and H. Schüze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[12] C. McDiarmid. On the method of bounded differences. *Surveys in Combinatorics*, 1989.

[13] N. Palatin, A. Leizarowitz, A. Schuster, and R. Wolff. Mining for misconfigured machines in grid systems. In *Proceedings of KDD*, 2006.

[14] 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.

[15] R. H. Randles. A distribution-free multivariate sign test based on interdirections. *Journal of the American Statistical Association*, 1989.

[16] R. Serfling and S. Mazumder. Exponential probability inequality and convergence results for the median absolute deviation and its modifications. *Statistics & Probability Letters*, 2009.

[17] J. Tukey. Mathematics and picturing data. In *Proceedings of the ICM*, 1975.

[18] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox. Ensembles of models for automated diagnosis of system performance problems. In *Proceedings of DSN*, 2005.