Unsupervised Anomaly Detection in Large Datacenters

Moshe Gabel

Unsupervised Anomaly Detection in Large Datacenters

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Moshe Gabel

Submitted to the Senate
of the Technion — Israel Institute of Technology
Sivan 5773 Haifa June 2013

This research was carried out under the supervision of Prof. Assaf Schuster in the Faculty of Computer Science, and Dr. Ran Gilad-Bachrach in Microsoft Research.

Some results in this thesis have been published during the course of the author's research period in a conference article by the author and research collaborators, the most up-to-date version of which being:

> M. Gabel, A. Schuster, R.-G. Bachrach, and N. Bjorner. Latent fault detection in large scale services. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–12, 2012.

Acknowledgements

I would like to express my gratitude to my advisors, Prof. Assaf Schuster and Dr. Ran Gilad-Bachrach, for their continuous instruction and patience. I have learned much from them, and where I did not – the fault surely lies with me. Their invaluable guidance and their persistent encouragement over the years has made this work possible.

I would also like to thank our co-author, Dr. Nikolaj Bjørner, for his insightful suggestions and good humor at stressful times.

To my friends, who let me practice conference talks on them, or even worse – thank you! You know who your are.

Finally, a special thank you for my family. Without your constant and uncompromising support I would not have made it to the finish line.

The generous financial help of the Technion is gratefully acknowledged.

Contents

List of Figures

List of Tables

Abstract

Unexpected machine failures, with their resulting service outages and data loss, pose challenges to datacenter management. Complex online services run on top of datacenters that often contain thousands of machines. With so many machines, failures are common, and automatic monitoring is essential.

Many existing failure detection techniques do not adapt well to the unpredictable and dynamic environment of large-scale online services. They rely on static rules, obsolete historical logs or costly (often unavailable) training data. More flexible techniques are impractical, as they require on deep domain knowledge, unavailable 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 on large real-world services, in which over 20% of machine failures were preceded by such *latent faults*.

We propose a proactive approach to failure prevention by detecting performance anomalies without prior knowledge about the monitored service. We present a novel framework for statistical latent fault detection using only ordinary machine counters collected as standard practice. The main assumption in our framework is that that at any point in time, most machines function well. By comparing machines to each other, we can then find those machines that exhibit latent faults.

We demonstrate three detection methods within the framework, and apply them to several real-world production services. The derived tests are domain-independent and unsupervised, require neither background information nor parameter tuning, and scale to very large services. We prove strong guarantees on the false positive rates of our tests, and show how they hold in practice.

Abbreviations and Notations

$\mathcal{M}$	:	the set of all tested machines
$\mathcal{C}$	:	the set of all counters selected by the pre-processing algorithm
$\mathcal{C}$	:	the set of all counters available in the system (before pre-processing)
$\mathcal{T}$	:	the set of time points when counters were sampled during preprocessing
m, m'	:	specific machines, $m, m' \in \mathcal{M}$
c, c'	:	specific counters, $c, c' \in \mathcal{C}$
t, t'	:	specific times, $t, t' \in \mathcal{T}$
M	:	the number of machines, $M = \lvert\mathcal{M}\rvert$
C	:	the number of counters selected by the preprocessing algorithm, $C = \lvert\mathcal{C}\rvert$
T	:	the number of times points where counters were sampled, $T = \lvert\mathcal{T}\rvert$
$n_c(m)$	:	the number of times machine m reported the counter c
$z_c(m,t)$	:	the last value of counter c on machine m before time t
$x(m,t)$	:	vector of C preprocessed counter values for machine m at time t
$x_c(m,t)$	:	the value of preprocessed counter c on machine m at time t
$x(t)$	:	vectors at time t for all M machines, $x(t) = \{x(m,t) \lvert m \in \mathcal{M}\}$
$S(m, x(t))$	:	test function assigning score vector (or scalar) to machine m at time t
v_m	:	score vector or scalar for machine m
mean	:	empirical mean, $\text{mean}_{i\in S}(X_i) = \frac{1}{\lvert S\rvert}\sum_{i\in S} X_i$
STD	:	standard deviation, $\text{STD}_{i\in S}(X_i) = \sqrt{\frac{1}{\lvert S\rvert}\sum_{i\in S}\left(X_i - \text{mean}_{i\in S}(X_i)\right)^2}$
$p_{90}(S)$	:	the 90^{th}-percentile of S
MAD	:	Median Absolute Deviation
LOF	:	Local Outlier Factor

Chapter 1

Introduction

In recent years the demand for computing power and storage has increased. Modern web services and clouds rely on large datacenters, often comprised of thousands of machines [Ham07]. For such large services, it is unreasonable to assume that all machines are working properly and are well configured [PLSW06, NDO11].

Monitoring is essential in datacenters, since unnoticed faults might accumulate to the point where redundancy and fail-over mechanisms break. Yet the large number of machines in datacenters makes manual monitoring impractical. Instead machines are usually monitored by collecting and analyzing performance counters [BGF+10, CJY07, Isa07, SOR+03]. Hundreds of counters per machine are reported by the various service layers, from service-specific metrics (such as the number of queries for a database) to general metrics (such as CPU utilization).

This work describes a statistical framework for detecting *latent faults* – performance anomalies that indicate a fault, or could eventually result in a fault. Our method does not require historical data, nor background knowledge about the monitored service. It adapts to changes in workload and monitored service, and provides statistical guarantees on the rate of false positives.

Our experiments provide evidence that latent faults are common even in well-managed datacenters. We show that these faults can be detected days in advance with high precision, without extensive knowledge of the service, learning from historical logs, or tuning the mechanism to the specific system. This enables a proactive approach [NM07]: machine failures can be predicted and handled effectively and automatically without service outages or data loss.

1.1 Background: Monitoring and Fault Detection

The challenge of monitoring stems from the unpredictable and dynamic environment that large-scale online services must live in. First, workload is often difficult to predict and is constantly changing. Second, software (and sometimes hardware) is frequently updated, changing the service's baseline behavior. Third, obtaining expertly-labeled

historical data is expensive, since it requires manual inspection of the data by people with extensive knowledge of the service and deep insight into how it works. Finally, false alarms can be costly since they involve support engineers responding to the alarm. Ubiquitous false alarms also give rise to "alarm fatigue", where personnel start ignoring alarms since most of them are false.

Most existing failure detection techniques are inflexible – they do not adapt to the frequent changes in the monitored service and its environment.

Rule-based Monitoring

Existing automated systems for detecting failures are mostly rule-based. A set of watchdogs [Isa07, NM07] 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.

Learning From the Past

Others have noticed the shortcomings of these rule-based approaches. More advanced methods model service behavior from historical logs. Supervised machine learning approaches [BGF+10, CZL+04, CGKS04, PBYH+08, SOR+03] propose training a detector on historic annotated data. Others [CJY07, BLB+10] analyze logs from periods from when the service is guaranteed to be healthy to extract model parameters.

Such approaches can fall short because they are sensitive to deviations in workloads and changes in the monitored service itself [ZCG+05, HCSA07]. After such changes the historical logs and the learned model are no longer relevant. These approaches can also be expensive since they require labeled data for operations, and re-labeling when the service changes. Obtaining this labeled data can be difficult.

Console Log Analysis

More flexible, unsupervised approaches to failure detection have been proposed for high performance computing (HPC). Typical approaches [OAS08, LFY+10, XHF+09] 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.

Domain Specific and Other Approaches

Some approaches [KTGN10, KGN08] are unsupervised and flexible, but are not domain independent. They make use of domain insights and knowledge into monitored service, for example in the domain of distributed file systems, and are therefore limited to specific systems. Others have proposed injecting code into the monitored service to periodically examine it [PLSW06]. This approach is intrusive and hence prohibitive in many cases.

1.2 This Work: Early Detection of Faults

Recent approaches to the monitoring problem [KGN08, KDJ+12] focus on early detection and handling of performance problems, or *latent faults*. Latent faults are machine behaviors that are indicative of a fault, or could eventually result in a fault, yet fly under the radar of monitoring systems because they are not acute enough, or were not anticipated by the monitoring system designers.

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 general framework and methods that address all these issues simultaneously using aggregated numerical counters normally collected by datacenters.

We focus on detecting anomalous machine behavior – latent faults. 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 (Section 4.5).

We develop a domain independent framework for identifying latent faults (Chapter 2). 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 [KTGN10, OAS08]. 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 (Chapter 3). 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 (Chapter 4).

Our technique is agile: we demonstrate its ability to work efficiently on different services with no need for tuning, yet with a guaranteed 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.

1.3 Related Work

The problem of automatic machine failure detection was studied by several researchers in recent years, and the techniques they proposehave so far relied on historical knowledge, domain-specific insights, or textual console logs.

Approaches That Rely on Historical Data

These approaches model the service behavior based on historical logs, often using supervised machine learning. Such approaches are not flexible to changes in workload or the monitored system, and can also require expertly labeled data that is difficult to obtain.

Chen et al. [CZL+04] presented a supervised approach based on learning decision trees, and successfully applied it to a large real-world service. The system requires labeled examples of failures and domain knowledge. Moreover, supervised approaches are less adaptive to workload variations and to platform changes. Cohen et al. [CGKS04] induce a tree-augmented Bayesian network classifier. Although this approach does not require domain knowledge other than a labeled training set, the resulting classifier is sensitive to changing workloads. Ensembles of models are used in [ZCG+05] to reduce the sensitivity of the former approach to workload changes, at the cost of decreased accuracy when there are too many failure types ([HCSA07]). Sahoo et al. [SOR+03] 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. Pelleg et al. [PBYH+08] explore failure detection in virtual machines using decision trees on a set of 6 manually selected hypervisor counters.

Though the basis is domain independent, the system is supervised, requiring training on labeled examples and manually selected counters.

Chen et al. [CJY07] separate metrics into workload counters and internal measurements. They analyze the correlation between these sets of metrics and track them over time. This approach requires training the system to model baseline correlations. It also requires domain knowledge when choosing counters. Bronevetsky et al. [BLB+10] 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. Bodík et al. [BGF+10] produce fingerprints from aggregate counters that describe the state of the entire datacenter, and use these fingerprints to identify system crises, points in time where the system performance falls below acceptable values. As with other supervised techniques, the approach requires labeled examples and is sensitive to changes in the monitored system or workload. The authors present quick detection of system failures that have already occurred, whereas we focus on detection of latent faults ahead of machine failures.

Approaches Requiring No Historical Data

Palatin et al. [PLSW06] 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. [KTGN10] 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.

Kavulya et al. [KGN08] present Gumshoe, which detects performance problems in replicated file systems, and is the most similar to our work. As we do, they assume that the system is comprised of homogenous machines, most of which are error free. Moreover, they compare machines using carefully selected performance metrics, and their method detects anomalies while still being workload invariant. Metrics of different machines are assumed to be correlated to each other, either in their raw counter form or a summary time series that represents their changes. Similar to [KTGN10], their work only considers replicated file systems, and uses a small set of manually selected counters appropriate to such systems. Unlike this work, their algorithm requires tuning parameters for each system, and it is sensitive to the choice of parameters. Finally, they

do not provide statistical guarantees on performance.

Textual Console Log Analysis

There are several unsupervised textual console log analysis methods. Oliner et al. [OAS08] 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. [XHF+09] analyze source code to parse console log messages and use principal component analysis to identify unusual message patterns. Lou et al. [LFY+10] represent code flow by identifying linear relationships in counts of console log messages. Unlike [OAS08, LFY+10], our method has a 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 log analysis is infeasible in large-scale services with high transaction volume.

Chapter 2

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 [KTGN10, KGN08, OAS08], 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.

2.1 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 4.5 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 point $t \in \mathcal{T}$. 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 2.2), the framework incorporates pluggable *tests* (aka outlier detection methods) to compare machine operation. At any

time t, the input $x(t)$ to a test S consists of the vectors $x(m,t)$ for all machines at time t: $x(t) = \{x(m,t)|m \in \mathcal{M}$. 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.

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 2.3 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 2.2 (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 Chapter 3.

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' denote specific machines, and $M = |\mathcal{M}|$ denotes the number of machines. $\mathcal{C}$ is the set of all counters selected by preprocessing step (step 1), 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 $T = |\mathcal{T}|$. Let $x(m,t)$ be the vector of preprocessed counter values for machine m at time t, $x_c(m,t)$ the value of counter c for machine m at time t, and $x(t)$ the set of all counter values for all machines at time t. x and x' denote sets of inputs $x(m,t)$ and $x'(m,t)$, respectively, for all m and t. Finally, in the preprocessing section Section 2.2 $\mathcal{C}'$ denotes the set of all counters available in the system, i.e. before the preprocessing step.

Assumptions

In modeling the problem we make several reasonable assumptions (see, e.g., [KTGN10, KGN08, OAS08]) 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 Chapter 4 (except where otherwise noted).

- 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.

2.2 Preprocessing

Clearly, our model is simplified, and in practice not all of its assumptions about counters hold. Thus the importance of the preprocessing algorithm Algorithm 2.1: 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.

First, 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, as demonstrated in Figure 2.1; hence they are automatically detected by looking at the variability of the reporting rate and are removed by the preprocessing.

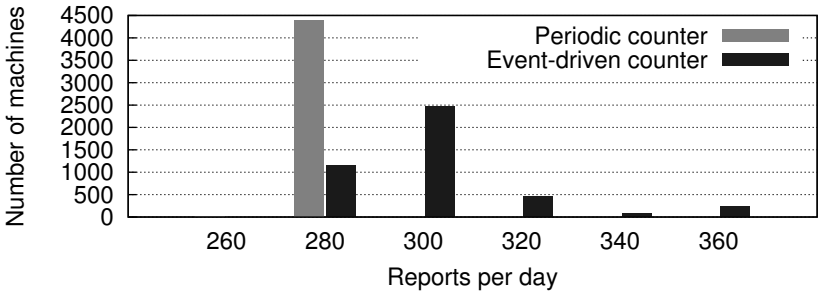


Figure 2.1: Histogram of number of reports for two kinds of counters. The report rate across different machines of the event-driven counter has higher variance.

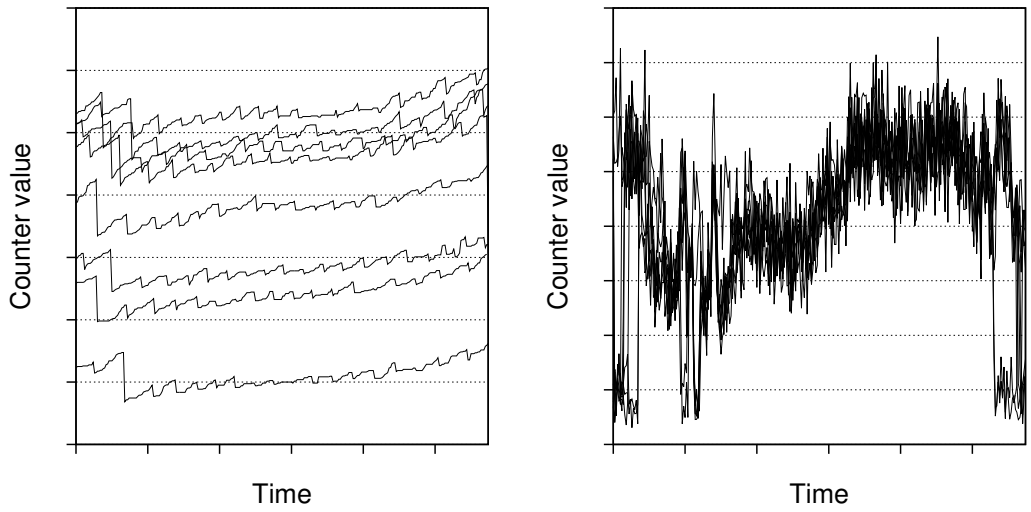
For each counter c and machine m, let $n_c(m)$ be the number of times machine m reported the counter c during the test period. We expect all machines to have similar number of reports for a periodic counters. To robustly detect the typical number of reports for the counter c we compute the median number of reports, denoted by n_c. This median is used to compute a robust notion of the variability in the number of reports, which is similar to standard deviation: the *Median Absolute Deviation* (MAD) (see, e.g., [SM09]). The MAD of a sample S is defined as

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

n_c is also used to scale variability so that the same threshold can be used for frequent and infrequent counters. We therefore define the variability as 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 90^{th} percentile represents our assumption that most of the machines (at least 90%) are working properly. We also eliminate infrequent counters: in our implementation, counters that are being reported less than 6 times a day are discarded.

After eliminating the non-periodic counters, preprocessing samples counter values at equal time intervals (5 minutes in our implementation), so that machines can be compared at those time points $t \in \mathcal{T}$. At this stage the counters are also normalized to have zero mean and a unit variance (across all machines and times), in order to eliminate artifacts of scaling and numerical instabilities. Counters with the same constant value across all machines are discarded.

Finally, 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. Due to our assumptions, we expect that counters will have similar means when measured on different machines (see, e.g. Figure 2.2). On



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

Figure 2.2: Counter values for 8 machines. The counter in 2.2a shows different means for individual machines. On the other hand, despite unpredictable variation over time, in 2.2b all machines act in tandem.

the other hand we cannot simply eliminate all counters with deviations from the mean, since we would have no counters left to detect malfunctioning machines. Additionally machines may skew mean and variance calculations since they are outliers.

Therefore, we use robust statistics in the following way. We first compute $\mu_c(m)$, the median of the value of the counter c on machine m, for every machine $m \in \mathcal{M}$. Let μ_c be the median of the median values: $\mu_c = \text{median}_{m \in \mathcal{M}}(\mu_c(m))$. As before, we use the median absolute deviation as a robust version of standard deviations. Hence we, compute the 90^{th} percentile of

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

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

The process of dropping counters is particularly important when monitoring virtual machines. It eliminates counters affected by "cross-talk" between virtual machines sharing 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 4.4.

The preprocessing algorithm Algorithm 2.1 has three parameters that need to be tuned. However, as demonstrated in Section 4.7 the same parameter values can be used for very different services, and furthermore our tests are robust to moderate changes in preprocessing parameter values: such changes do not affect performance in any significant way.

Algorithm 2.1 The preprocessing algorithm. Receives the raw counters, eliminates problematic counters and normalizes the data. $p_{90}(S)$ denotes the 90^{th}-percentile of S.

Let $\theta^1 = 0.01, \theta^2 = 2, \theta^n = 6$
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 counter c on m
for all counter $c \in \mathcal{C}'$ **do**
  $\nu_c \leftarrow \text{mean}_{m\in\mathcal{M},t\in\mathcal{T}}(z_c(m,t))$
  $\sigma_c \leftarrow \text{STD}_{m\in\mathcal{M},t\in\mathcal{T}}(z_c(m,t))$
  for all 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 for
  for all machine $m \in \mathcal{M}$ **do**
    $\mu_c(m) \leftarrow \text{median}_{t\in\mathcal{T}}(y_c(m,t))$
    $\text{mad}_c(m) \leftarrow \text{MAD}_{t\in\mathcal{T}}(y_c(m,t))$
  end for
  $n_c \leftarrow \text{median}_{m\in\mathcal{M}}(n_c(m))$
  $\psi_c^1 \leftarrow p_{90}\left(\left|\frac{n_c(m)-n_c}{n_c}\right|\right)$
  $\mu_c \leftarrow \text{median}_{m\in\mathcal{M}}(\mu_c(m))$
  $\psi_c^2 \leftarrow p_{90}\left(\left|\frac{\mu_c(m)-\mu_c}{\text{mad}_c(m)}\right|\right)$
  if $(\psi_c^1 \le \theta^1)$ **and** $(n_c \ge \theta^n)$ **and** $(\psi_c^2 \le \theta^2)$ **then**
    Add counter c to set of selected counters $\mathcal{C}$
    for all machine $m \in \mathcal{M}$ **and** time $t \in \mathcal{T}$ **do**
      $x_c(m,t) \leftarrow y_c(m,t)$
    end for
  else
    Discard counter c
  end if
end for

2.3 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, encapsulated as two lemmas.

Recall that that the framework defines the scoring function of a single machine as follows:

$$\|v_m\| = \left\| \frac{1}{T} \sum_{t \in \mathcal{T}} S(m, x(t)) \right\| \quad .$$

To compute a p-value for a machine, we compare $\|v_m\|$ to its expected value, or to its empirical mean if the expected value is unknown.

The first method assumes the expected value of the scoring function is known a priori 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 require that test functions be bounded as follows:

Definition 2.3.1. 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. $\|S(m, x(t)) - S(m, x'(t))\| \leq L_1$.

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.

For a test S that is L_1, L_2-bounded as in Definition 2.3.1, Lemma 2.3.2 and Lemma 2.3.2 below provide probability guarantees for the divergence of $\|v_m\|$ from its expected value or from its empirical mean, respectively.

Bounded Differences Inequality and Triangle Inequality

Our lemmas rely on the bounded differences inequality, as well as on the triangle inequality and its reverse.

The *independent bounded differences inequality* [McD89] gives a concentration result for functions of bounded random variables. It provides an upper bound on the probability that such a function deviates from its expected value:

Theorem ([McD89, Lemma 1.3])**.** *Let $X_1, \ldots, X_n$ be independent random variables, with X_k taking values in the set A_k for each k. Suppose that the measurable function $f : \Pi_{k=1}^n A_k \to \mathbb{R}$ satisfies*

$$\left| f(x) - f(x') \leq c_k \right|$$

whenever the vectors x and x' differ only in the k-th coordiate. Let Y be the random variable $f(X_1, \ldots, X_n)$. Then for any $t > 0$,

$$\Pr\left[Y - \mathrm{E}\left[Y\right] \geq t\right] \leq \exp\left(-\frac{2t^2}{\sum_{k=1}^n c_k^2}\right) \quad ,$$

$$\Pr\left[\mathrm{E}\left[Y\right] - Y \geq t\right] \leq \exp\left(-\frac{2t^2}{\sum_{k=1}^n c_k^2}\right) \quad ,$$

and

$$\Pr\left[\left|Y - \mathrm{E}\left[Y\right]\right| \geq t\right] \leq 2\exp\left(-\frac{2t^2}{\sum_{k=1}^n c_k^2}\right) \quad .$$

The triangle inequality is

$$\|x + y\| \leq \|x\| + \|y\| \quad .$$

Consequently, the reverse triangle inequality is

$$\left| \|x\| - \|y\| \right| \leq \|x - y\| \quad .$$

When the Expected Value $\mathrm{E}\left[\|v_m\|\right]$ is Known

When $\mathrm{E}\left[\|v_m\|\right]$ is known a priori, we use Lemma 2.3.2 to give an upper bound on the probability that the score of a healthy machine is above the expected value by some amount γ.

Lemma 2.3.2. *Consider a test S that is L_1, L_2- bounded as in Definition 2.3.1. Assume that $\forall m, t, \quad x(m,t) \sim X(t)$. Then for every $\gamma > 0$,*

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

Proof. We have T independent random variables $X(t)$, each taking values in $\mathbb{R}$. Our measurable function is

$$f(x) = \|v_m\| = \frac{1}{T}\left\|\sum_{t\in\mathcal{T}} S(m, x(t))\right\| \quad ,$$

and from the triangle inequality we can write:

$$\|v_m\| = \frac{1}{T}\left\|\sum_{t\in\mathcal{T}} S(m,x(t))\right\| \leq \frac{1}{T}\sum_{t\in\mathcal{T}}\|S(m,x(t))\| \quad . \tag{2.1}$$

For every time t', for any x and x' which differ only at one time t', then $\forall t \neq t', S(m,x(t)) = S(m,x'(t))$. By applying the triangle inequality, and since S is L_1, L_2-bounded we can write:

$$\begin{aligned}
\left|f(x) - f(x')\right| &= \left|\frac{1}{T}\left\|\sum_{t\in\mathcal{T}} S(m,x(t))\right\| - \frac{1}{T}\left\|\sum_{t\in\mathcal{T}} S(m,x'(t))\right\|\right| \\
&= \frac{1}{T}\left|\left\|\sum_{t\in\mathcal{T}} S(m,x(t))\right\| - \left\|\sum_{t\in\mathcal{T}} S(m,x'(t))\right\|\right| \\
&\leq \frac{1}{T}\left|\sum_{t\in\mathcal{T}}\|S(m,x(t))\| - \sum_{t\in\mathcal{T}}\left\|S(m,x'(t))\right\|\right| \\
&= \frac{1}{T}\left|\left\|S(m,x(t'))\right\| - \left\|S(m,x'(t'))\right\|\right| \\
&\leq \frac{1}{T}\left\|S(m,x(t')) - S(m,x'(t'))\right\| \\
&\leq \frac{L_1}{T} \quad .
\end{aligned} \tag{2.2}$$

We can now apply the bounded differences inequality Theorem ([McD89, Lemma 1.3]) to derive a bound for one specific machine m:

$$\begin{aligned}
\Pr\left[\|v_m\| - \mathrm{E}\left[\|v_m\|\right] \geq \gamma\right] &\leq \exp\left(-\frac{2\gamma^2}{\sum_{t\in\mathcal{T}}\frac{L_1^2}{T^2}}\right) \\
&= \exp\left(-\frac{2\gamma^2}{T\frac{L_1^2}{T^2}}\right) \\
&= \exp\left(-\frac{2T\gamma^2}{L_1^2}\right) \quad .
\end{aligned} \tag{2.3}$$

Finally, we apply the union bound $\Pr\left[\cup_i A_i\right] \leq \sum_i \Pr\left(A_i\right)$ to (2.3) to arrive at a bound for all machines:

$$\begin{aligned}
\Pr\left[\exists m \text{ s.t. } \|v_m\| \geq \mathrm{E}\left[\|v_m\|\right] + \gamma\right] &\leq \sum_{m\in\mathcal{M}} \Pr\left[\|v_m\| - \mathrm{E}\left[\|v_m\|\right] \geq \gamma\right] \\
&\leq \sum_{m\in\mathcal{M}} \exp\left(-\frac{2T\gamma^2}{L_1^2}\right) \\
&= M\exp\left(-\frac{2T\gamma^2}{L_1^2}\right)
\end{aligned}$$

which completes the proof. □

When the Expected Value $\mathrm{E}[\|v_m\|]$ is Unknown

When $\mathrm{E}[\|v_m\|]$ is not known a priori, we use the empirical mean as a proxy for the true expected value. Lemma 2.3.3 gives upper bound on the probability that the score of a healthy machine is above the empirical mean value by some amount γ.

Lemma 2.3.3. *Consider a test S that is L_1, L_2- bounded as in Definition 2.3.1. Assume that $\forall m, t, \quad x(m,t) \sim X(t)$, and that $\forall m, m'$, $\mathrm{E}[\|v_m\|] = \mathrm{E}[\|v_{m'}\|]$. Denote by $\hat{v}$ the empirical mean of v_m:*

$$\hat{v} = \frac{1}{M} \sum_{m \in \mathcal{M}} \|v_m\| \quad .$$

Then for every $\gamma > 0$,

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

Proof. We will first limit the divergence of the empirical mean from the true expectation, using the bounded differences inequality.

The empirical $\hat{v}$ is itself a function of MT random variables, $\|v_m(x(t))\|$ for every $m \in \mathcal{M}, t \in \mathcal{T}$:

$$\hat{v} = \hat{v}(x) = \frac{1}{M} \sum_{m \in \mathcal{M}} \|v_m(x(t))\| = \frac{1}{MT} \sum_{m \in \mathcal{M}} \left\| \sum_{t \in \mathcal{T}} S(m, x(t)) \right\| \quad .$$

Let x, x' differ only in a single coordinate (m', t'), meaning at time t' for machine m'. Then since S is L_1, L_2-bounded, we can show that

$$\left| \hat{v}(x) - \hat{v}(x') \right| \leq \frac{L_1 + L_2(M-1)}{MT} \quad .$$

This is because $|\|v_m(x)\| - \|v_m(x')\||$ is upper bounded by $\frac{L_1}{T}$ when $m = m'$, and by $\frac{L_2}{T}$ for every $m \neq m'$. Formally, from (2.2) and with similar steps, we use the triangle

inequality and the fact that $\forall t \neq t', S(m, x(t)) = S(m, x'(t))$ to derive the bound:

$$\begin{aligned}
\left|\hat{v}(x) - \hat{v}(x')\right| &= \left| \frac{1}{M} \sum_{m \in \mathcal{M}} \|v_m(x)\| - \frac{1}{M} \sum_{m \in \mathcal{M}} \|v_m(x')\| \right| \\
&= \frac{1}{M} \left| \sum_{m \in \mathcal{M}} \|v_m(x)\| - \|v_m(x')\| \right| \\
&= \underbrace{\frac{1}{M} \left| \|v_m(x)\| - \|v_m(x')\| \right|}_{m=m'} + \underbrace{\frac{1}{M} \left| \sum_{m \neq m'} \|v_m(x)\| - \|v_m(x')\| \right|}_{m \neq m'} \\
&\leq \frac{L_1}{MT} + \frac{1}{MT} \left| \sum_{m \neq m'} \left\| \sum_{t \in \mathcal{T}} S(m, x(t)) - S(m, x'(t)) \right\| \right| \\
&= \frac{L_1}{MT} + \frac{1}{MT} \left| \sum_{m \neq m'} \left\| S(m, x(t')) - S(m, x'(t')) \right\| \right| \\
&\leq \frac{L_1}{MT} + \frac{L_2(M-1)}{MT} = \frac{L_1 + L_2(M-1)}{MT} \quad . \qquad (2.4)
\end{aligned}$$

We are now ready to apply the bounded differences inequality to $\hat{v}$:

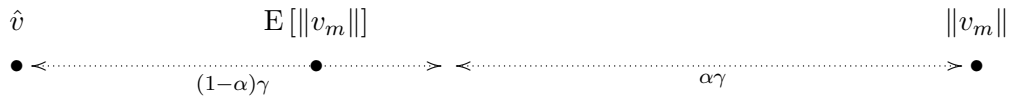
$$\begin{aligned}
\Pr\left[\hat{v} \leq \mathrm{E}\left[\hat{v}\right] - \gamma\right] &\leq \exp\left(-\frac{2\gamma^2}{\sum_{m \in \mathcal{M}, t \in \mathcal{T}} \left(\frac{L_1 + L_2(M-1)}{MT} \right)^2} \right) \\
&= \exp\left(-\frac{2MT\gamma^2}{\left(L_1 + L_2(M-1)\right)^2} \right) \quad . \qquad (2.5)
\end{aligned}$$

Now that we know how the empirical mean is concentrated around the true expected value, we will combine (2.5) with Lemma 2.3.3 to yield a bound for L_1, L_2-bound tests. In intuitive terms, we limit how far $\|v_m\|$ is allowed to stray from the true mean $\mathrm{E}\left[\|v_m\|\right]$, and how far the empirical mean $\hat{v}$ is allowed to stray from the true mean.

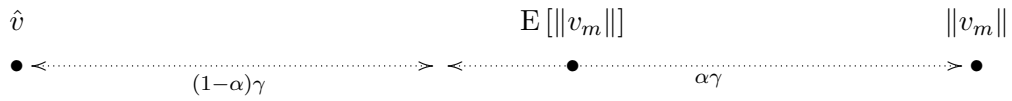
From linearity of expectation, and since $\forall m, m' : \mathrm{E}\left[\|v_m\|\right] = \mathrm{E}\left[\|v_{m'}\|\right]$, we know that the empirical mean of $\|v_m\|$ converges to the expectation:

$$\mathrm{E}\left[\hat{v}\right] = \mathrm{E}\left[\frac{1}{M} \sum_{m \in \mathcal{M}} \|v_m\| \right] = \frac{1}{M} \sum_{m \in \mathcal{M}} \mathrm{E}\left[\|v_m\|\right] = \mathrm{E}\left[\|v_m\|\right] \quad .$$

We now combine (2.5) with Lemma 2.3.3 using an affine combination parameter $0 < \alpha < 1$, which we will set later. If there exists m such that $\|v_m\| - \hat{v} \geq \gamma$, we use α to restrict where $\|v_m\|$ can be in relation to the empirical mean and the expected value (the true mean). One or both of the following must hold: either $\|v_m\| - \mathrm{E}\left[\|v_m\|\right] \geq \alpha\gamma$ (meaning $\|v_m\|$ is far enough above its expected value), or $\hat{v} \leq \mathrm{E}\left[\|v_m\|\right] - (1-\alpha)\gamma$ (meaning the empirical mean is far enough above the expected value). For the case where $\mathrm{E}\left[\|v_m\|\right] < \hat{v} < \|v_m\|$, it is trivially so. Figure 2.3 illustrates the case where



(a) Expected value is below split.



(b) Expected value is above split.

Figure 2.3: The case where $\|v_m\| - \hat{v} \geq \gamma$, and $\hat{v} < \mathrm{E}[\|v_m\|] < \|v_m\|$.

$\hat{v} < \mathrm{E}[\|v_m\|] < \|v_m\|$. Since the expected value $\mathrm{E}[\|v_m\|]$ is between the empirical mean and $\|v_m\|$, either it is below the γ split point and therefore $\|v_m\|$ is far enough above the expected value (Figure 2.3a), or $\mathrm{E}[\|v_m\|]$ is above the split point and therefore far enough above the empirical mean $\hat{v}$ (Figure 2.3b).

We can therefore state, combining Lemma 2.3.3 with (2.5):

$$\begin{aligned}
&\Pr[\exists m \text{ s.t. } \|v_m\| \geq \hat{v} + \gamma] \\
&\quad \leq \Pr[\exists m \text{ s.t. } \|v_m\| \geq \mathrm{E}[\|v_m\|] + \alpha\gamma] \\
&\qquad + \Pr[\hat{v} \leq \mathrm{E}[\hat{v}] - (1-\alpha)\gamma] \\
&\quad \leq M \exp\left(-\frac{2T\alpha^2\gamma^2}{L_1^2}\right) + \exp\left(-\frac{2MT(1-\alpha)^2\gamma^2}{(L_1 + L_2(M-1))^2}\right) \quad .
\end{aligned}$$

Finally, choosing

$$\alpha = \frac{\sqrt{M}L_1}{L_1\left(1+\sqrt{M}\right) + L_2\left(M-1\right)}$$

and substituting

$$
\begin{aligned}
&\Pr\left[\exists m \text{ s.t. } \|v_m\| \geq \hat{v} + \gamma\right] \\
&\quad \leq M \exp\left(-\frac{2T\alpha^2\gamma^2}{L_1^2}\right) + \exp\left(-\frac{2MT(1-\alpha)^2\gamma^2}{\left(L_1 + L_2(M-1)\right)^2}\right) \\
&\quad = M \exp\left(-\frac{2T\left(\frac{ML_1^2}{\left(L_1\left(1+\sqrt{M}\right)+L_2(M-1)\right)^2}\right)\gamma^2}{L_1^2}\right) \\
&\qquad + \exp\left(-\frac{2MT\left(\frac{\left(L_1\left(1+\sqrt{M}\right)+L_2(M-1)\right)-\sqrt{M}L_1}{\left(L_1\left(1+\sqrt{M}\right)+L_2(M-1)\right)}\right)^2\gamma^2}{\left(L_1 + (M-1)L_2\right)^2}\right) \\
&\quad = M \exp\left(-\frac{2MT\gamma^2}{\left(L_1\left(1+\sqrt{M}\right) + L_2\left(M-1\right)\right)^2}\right) \\
&\qquad + \exp\left(-\frac{2MT\gamma^2}{\left(L_1\left(1+\sqrt{M}\right) + L_2\left(M-1\right)\right)^2}\frac{\left(L_1 + L_2\left(M-1\right)\right)^2}{\left(L_1 + L_2(M-1)\right)^2}\right) \\
&\quad = (M+1) \exp\left(-\frac{2MT\gamma^2}{\left(L_1\left(1+\sqrt{M}\right) + L_2\left(M-1\right)\right)^2}\right)
\end{aligned}
$$

yields the stated result. □

Chapter 3

Derived Tests

Using the general framework described in Chapter 2, we describe three test implementations: the *sign test* (Section 3.1), the *Tukey test* (Section 3.2), and the *LOF test* (Section 3.3). Their analyses provide examples of the use of the machinery developed in Section 2.3. 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.

3.1 The Sign Test

The sign test [DM46] is a classic statistical test. It verifies the hypothesis that two samples share a common median. It has been extended to the multivariate case [Ran89]. We extend it to allow the simultaneous comparison of multiple machines. The "sign" of a machine m at time t is the average direction of its vector $x(m,t)$ to all other machines' vectors, and its score v_m is the sum of all these directions, divided by T.

The intuition is that healthy machines are similar on average, and any differences are random. Average directions are therefore random and tend to cancel each other out when added together, meaning v_m will be a relatively short vector for healthy machines. Conversely, if m has a latent fault then some of its metrics are consistently different from healthy machines, and so the average directions are similar in some dimensions. When summing up these average directions, these similarities reinforce each other and therefore v_m tends to be a longer vector.

Formally, let 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

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

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

for all 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 for
$\hat{v} \leftarrow \frac{1}{M} \sum_m \|v_m\|$
for all 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 if
end for

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 3.1. *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 3.1. 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 since

$$\left\| \frac{x(m,t) - x(m',t)}{\|x(m,t) - x(m',t)\|} \right\| \leq 1 \quad .$$

Applying Lemma 2.3.3, we obtain the stated result. □

Theorem 3.1 proves the correctness of the p-values computed by the sign test. For an appropriate significance level α, Theorem 3.1 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 [BGF+10]. It can also be used as a starting point for root cause analysis, which is a subject for future research.

3.2 The Tukey Test

The Tukey test is based on a different statistical tool, the Tukey depth function [Tuk75]. 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:

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

Algorithm 3.2 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 $R^C \rightarrow R^2$
  for all time $t \in \mathcal{T}$ **do**
    for all machine $m \in \mathcal{M}$ **do**
      $d(i, m, t) \leftarrow \mathsf{Depth}\left(\pi_i(x(m, t))|x(t)\right)$
    end for
  end for
end for
for all 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 for
$\hat{v} \leftarrow \frac{1}{M} \sum_m v_m$
for all 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 if
end for

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, it means that m tends to be outside the sample of points – an outlier. Hence 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 [Cha04]. Therefore, similarly to [CANR08], 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 $O(M \log(M))$, 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:

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

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 3.2. *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 3.2. 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 $0 \leq d(i,m,t) \leq \left\lceil \frac{M-1}{2} \right\rceil$ [Cha04]. Applying Lemma 2.3.3 with $-v_m$ and $-\hat{v}$, we obtain the stated result. □

3.3 The LOF Test

The LOF test is based on the *Local Outlier Factor* (LOF) algorithm [BKNS00], which is a popular outlier detection algorithm. The LOF test attempts to find outliers by comparing density of local neighborhoods. Vectors of faulty machines will often be dissimilar to those of healthy machines, and therefore end up with lower local density than their neighbours.

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

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

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.

Algorithm 3.3 The LOF test. Output a list of suspicious machines with p-value below significance level α.

for all time $t \in \mathcal{T}$ **do**
  $l(m,t) \leftarrow$ LOF of $x(m,t)$ in $x(t)$
  for all 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 for
end for
for all machine $m \in \mathcal{M}$ **do**
  $v_m \leftarrow \frac{1}{T}\sum_t S(m, x(t))$
  $\gamma \leftarrow \max(0, v_m - 1)$
  $p(m) \leftarrow M \exp\left(-\frac{T\gamma^2}{2}\right)$
  if $p(m) \leq \alpha$ **then**
    Report machine m as suspicious
  end if
end for

Theorem 3.3. *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 3.3. 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 $0 \leq r(m, x(t)) \leq M-1$. Moreover, under the assumption of this theorem, the expected value of the score is 1. Applying Lemma 2.3.2, we obtain the stated result. □

Chapter 4

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. Work distribution in the PE and SE services is static, rather than dynamic. Requests are sent to machines based on the key, rather than the current load on the service. 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 [Isa07]. We use the automatic

Table 4.1: Summary of terms used in evaluation.

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]

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.

4.1 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 historical data of the SE service, then used for all services. 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 4.5 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 [MRS08], 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 denote the fraction of healthy machines out of all suspicious machines. There is an inherent tradeoff between recall, precision and false positive rate. For example, if we flag all machines, recall is perfect (since all failing machines are flagged), but precision is very low (since only a fraction of flagged machines actually failed). Similarly, false positive rate can made 0 simply by never flagging any machine, but then recall is also 0. Table 4.1 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.

4.2 The LG Service

Table 4.2 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 4.2 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 4.1) 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 4.2, 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.

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

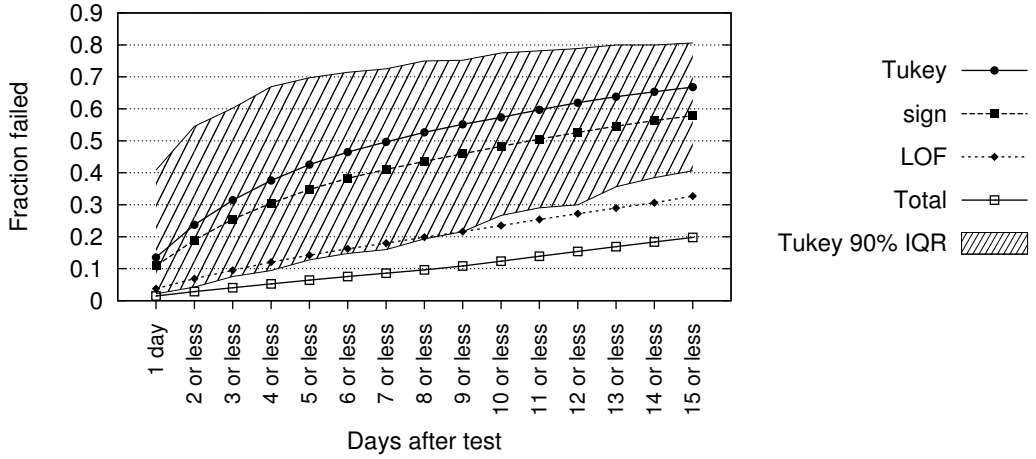


Figure 4.1: Cumulative failures on LG service, with the 5%-95% inter-quantile range of single day results for the best performing test, the Tukey test. Most of the faults detected by the sign test and the Tukey test become failures several days after detection.

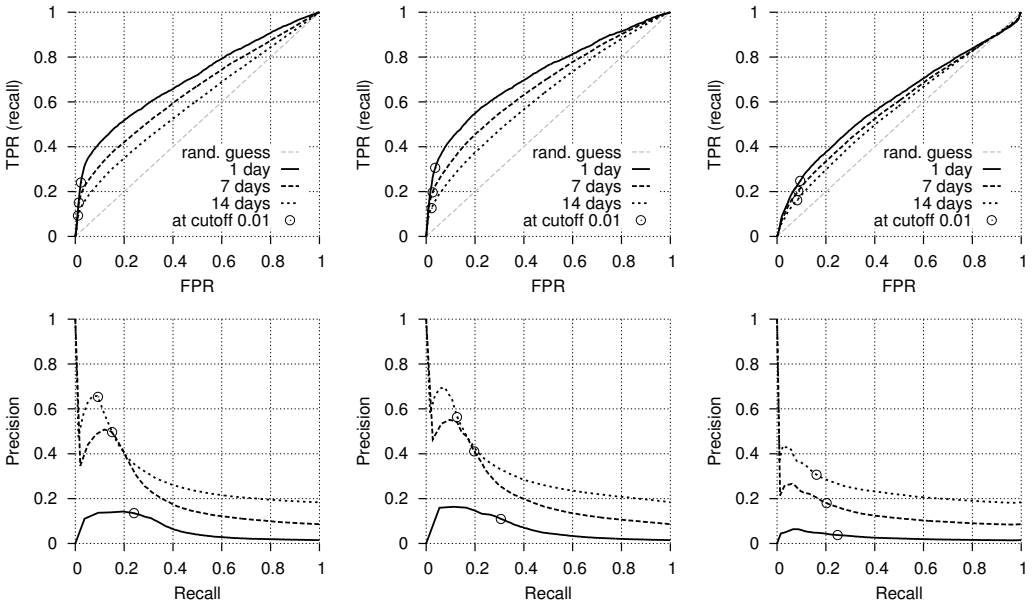
Table 4.2: Prediction performance on LG with significance level of 0.01. Numbers in parenthesis show the 5%-95% inter-quantile range of single day results.

Period	Test	Recall	FPR	Precision
1 day	Tukey	0.240 *(0.041–0.660)*	**0.023** *(0.012–0.035)*	**0.135** *(0.021–0.408)*
	sign	**0.306** *(0.100–0.717)*	0.037 *(0.025–0.065)*	0.109 *(0.015–0.308)*
	LOF	0.248 *(0.132–0.500)*	0.095 *(0.065–0.127)*	0.038 *(0.015–0.079)*
7 days	Tukey	0.151 *(0.025–0.293)*	**0.014** *(0.008–0.027)*	**0.497** *(0.160–0.725)*
	sign	**0.196** *(0.065–0.322)*	0.026 *(0.014–0.055)*	0.411 *(0.188–0.660)*
	LOF	**0.203** *(0.130–0.336)*	0.087 *(0.060–0.122)*	0.180 *(0.119–0.307)*
14 days	Tukey	0.093 *(0.022–0.192)*	**0.011** *(0.006–0.020)*	**0.653** *(0.385–0.800)*
	sign	0.126 *(0.053–0.241)*	0.022 *(0.011–0.049)*	0.563 *(0.389–0.741)*
	LOF	**0.162** *(0.091–0.273)*	0.082 *(0.055–0.118)*	0.306 *(0.218–0.605)*

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 4.3). Note that the weekly cycle does not affect



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

Figure 4.2: ROC and P-R curves on LG service. Highlighted points are for significance level $\alpha = 0.01$.

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.

4.3 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, 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 4.3.

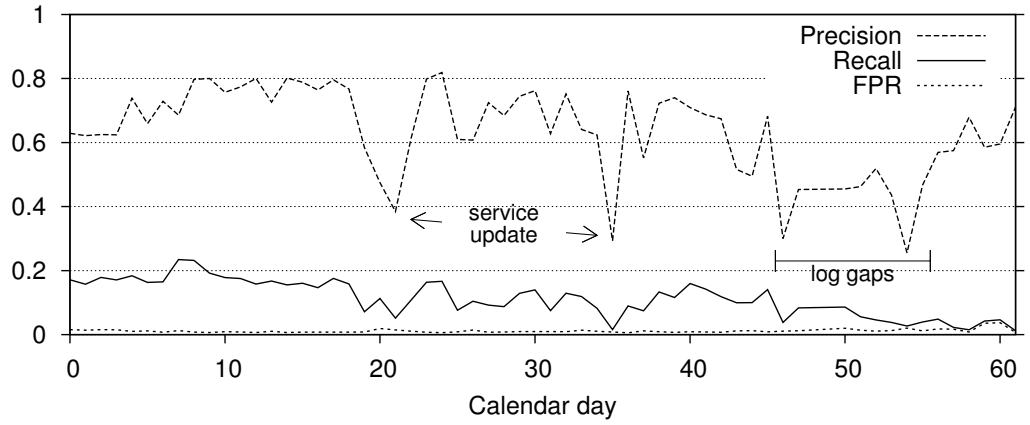
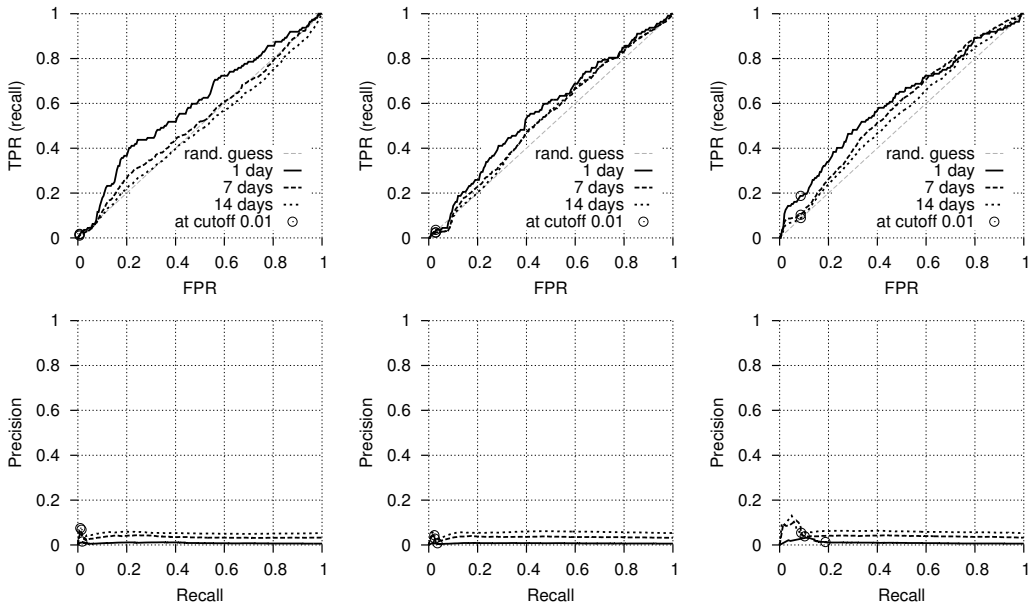


Figure 4.3: Tukey performance on LG across 60 days, with 14-day horizon. It shows the test is not affected by changes in the workload, quickly recovering from service updates on days 22 and 35. Lower performance on days 45–55 is an artifact of gaps in counter logs and updates on later days.



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

Figure 4.4: ROC and P-R curves on the SE service. Highlighted points are for significance level $\alpha = 0.01$.

Table 4.3: Prediction performance 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

ROC and P-R curves are in Figure 4.4. Our methods were able to detect and predict machine failures; 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, and despite the lack of dynamic load balancing, 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.

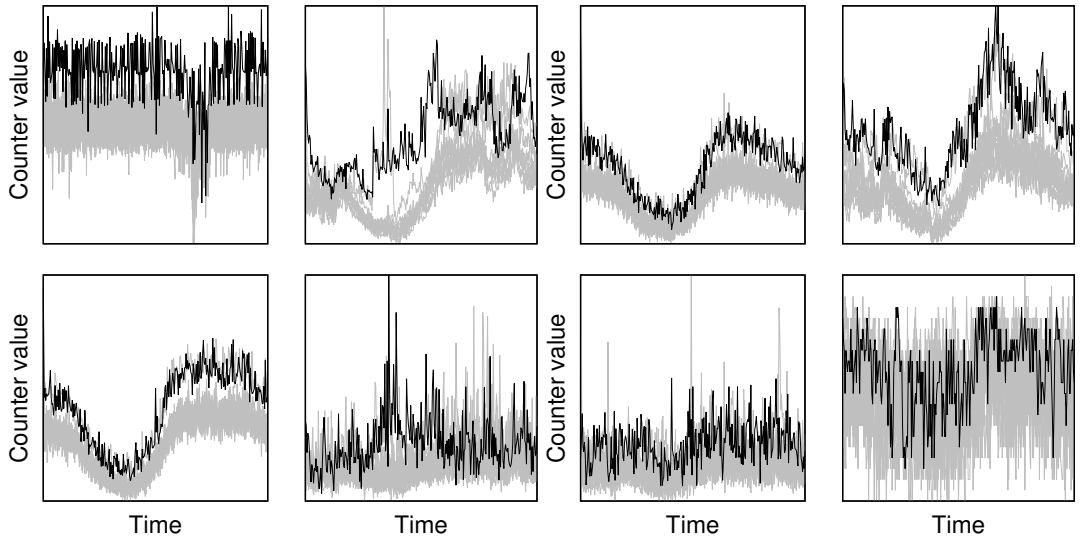
4.4 VM Service

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 overloaded. The relevant counters for this machine are plotted in Figure 4.5. 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.

4.5 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



(a) Detection day **(b)** Warning day

Figure 4.5: Aberrant counters for suspicious VM machine (black) compared to the counters of 14 other machines (gray).

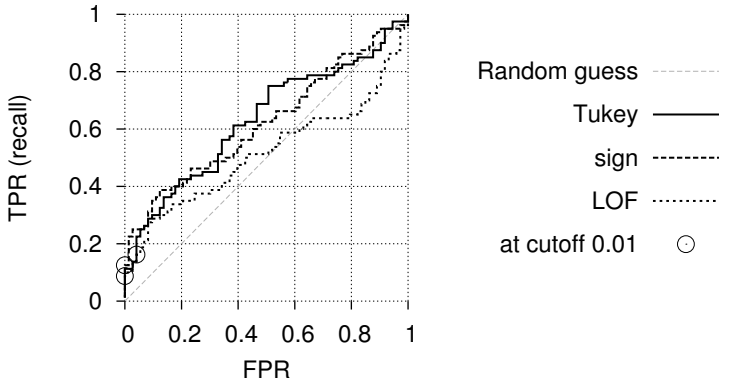


Figure 4.6: Detection performance on known failures in LG service. At least 20-25% of failures have preceding latent faults. Highlighted points are for $\alpha = 0.01$.

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 4.6 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 no 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 4.2 is close to the maximum possible.

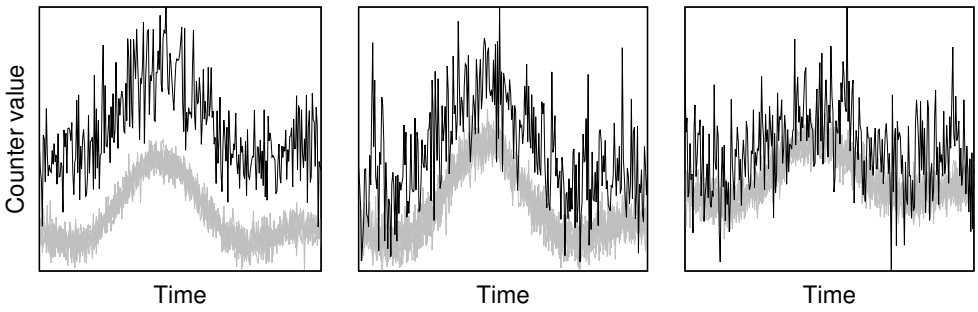


Figure 4.7: Three synthetic "counters" for 8 "machines". The highlighted machine (black) has a synthetic latent fault (aberrant counter behavior).

4.6 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.

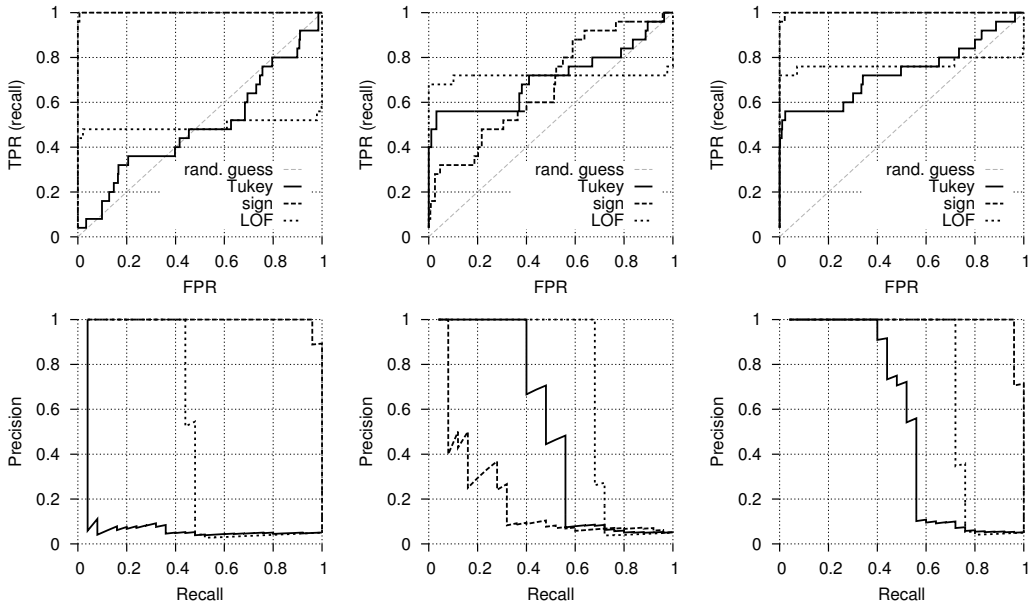
To better characterize the sensitivities of the different tests, we evaluated them on artificially generated data to which we injected three types of "faults": counter location (offset), counter scale, or both. We generated synthetic data equivalent to 500 machines, each reporting 150 counters every five minutes for a full day. We begin by generating varying workload with random noise at different scales. we define three types of "faults": location (offset), scale, or both (location+scale). 25 machines were selected as failing machines, and 10% of their counters are "faulty" with either different offset, scale or both. Figure 4.7 shows three such synthetic counters for several machines. 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 4.8. 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.

4.7 Filtering Counters in Preprocessing

As described in Section 2.2, the preprocessing stage removes some counters. Table 4.4 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



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

Figure 4.8: Performance on types of synthetic latent faults

Table 4.4: 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

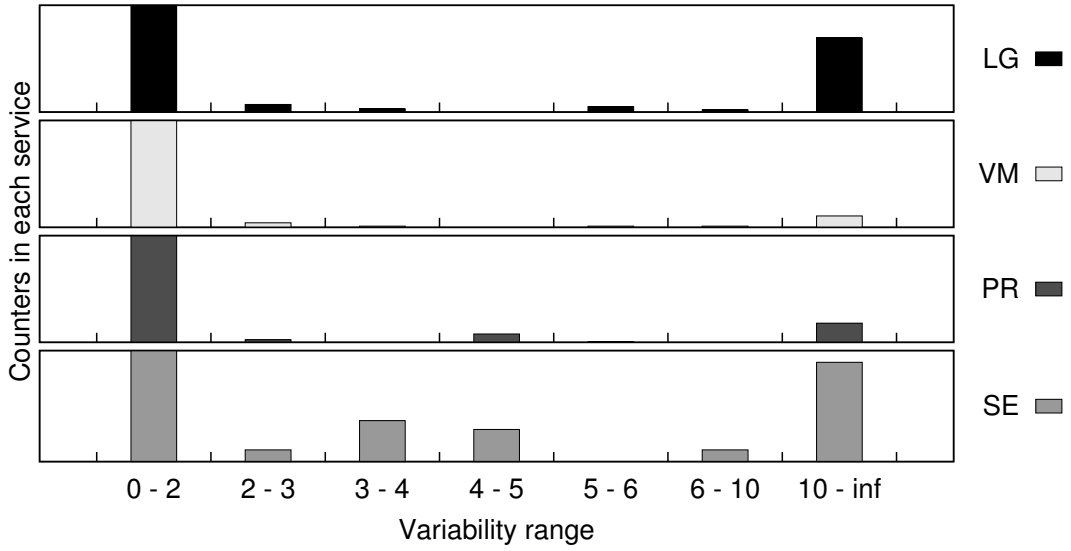


Figure 4.9: Histogram of counter mean variability for all services. The majority of counters have variability below 2.

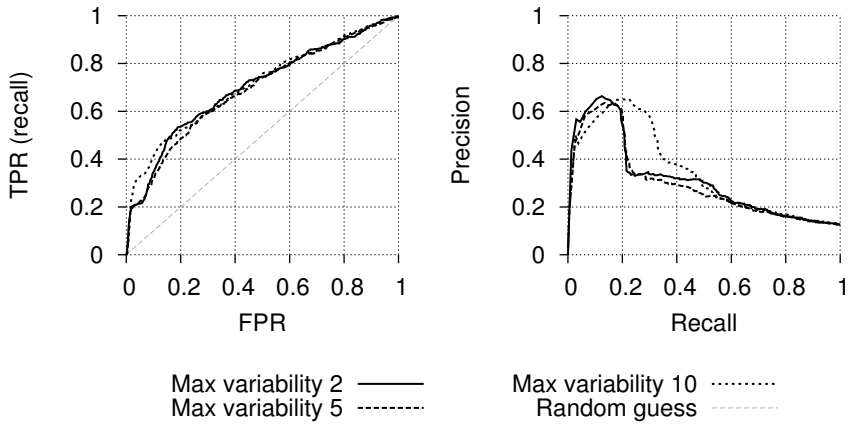


Figure 4.10: Sign test performance on one day of the LG service at difference mean variability thresholds.

counters. Figure 4.9 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. The results are shown in Figure 4.10. Performance is not very sensitive to the choice of threshold. With strict significance level, higher thresholds result in slightly better recall but slightly lower precision, confirming our expectations.

Chapter 5

Conclusion and Future Work

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. Our experiments show that latent faults are common even in well-managed datacenters, and that they can be detected well before they manifest as machine or service failure.

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.

There are several potential avenues for future work. First, the probability bounds offered by the framework could potentially be tightened using the empirical Bernstein bounds [AMS07, MSA08], since we often only know the empirical mean rather than the true mean. Also, variance can be bounded since input data is pre-scaled to variance of 1.0 during pre-processing. Second, the time window necessary for the tests is large, limiting their use in latency-sensitive scenarios. More sensitive tests and improved statistical framework can help overcome this issue. Additionally, one could extend the applicability of latent fault detection for services without dynamic load balancing. Rather than consider machine counter values, tests can monitor invariants unique to the work performed by the service. Finally, our framework assumes that machines are homogenous in hardware and software. While quite common in large scale online services, this is not always the case. Addressing this issue can make latent fault detection

applicable in many more scenarios.

In a larger context, the open question that remains 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 [NM07], 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 more reliable large-scale services.

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

Bibliography

[AMS07] J.-Y. Audibert, R. Munos, and C. Szepesvári. Tuning bandit algorithms in stochastic environments. In *Proc. ALT*, 2007.

[BGF+10] P. Bodík, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen. Fingerprinting the datacenter: Automated classification of performance crises. In *Proc. EuroSys*, 2010.

[BKNS00] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. *SIGMOD Rec.*, 2000.

[BLB+10] 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.

[CANR08] J. A. Cuesta-Albertos and A. Nieto-Reyes. The random Tukey depth. *Journal of Computational Statistics & Data Analysis*, 2008.

[CGKS04] 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.

[Cha04] T. M. Chan. An optimal randomized algorithm for maximum Tukey depth. In *Proc. SODA*, 2004.

[CJY07] H. Chen, G. Jiang, and K. Yoshihira. Failure detection in large-scale internet services by principal subspace mapping. *Trans. Knowl. Data Eng.*, 2007.

[CZL+04] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *Proc. ICAC*, 2004.

[DM46] W. J. Dixon and A. M. Mood. The statistical sign test. *Journal of the American Statistical Association*, 1946.

[Ham07] J. R. Hamilton. Architecture for modular data centers. In *Proc. CIDR*, 2007.

[HCSA07] C. Huang, I. Cohen, J. Symons, and T. Abdelzaher. Achieving scalable automated diagnosis of distributed systems performance problems. Technical report, HP Labs, 2007.

[Isa07] M. Isard. Autopilot: automatic data center management. *SIGOPS Oper. Syst. Rev.*, 2007.

[KDJ+12] S. Kavulya, S. Daniels, K. Joshi, M. Hiltunen, R. Gandhi, and P. Narasimhan. Draco: Statistical diagnosis of chronic problems in large distributed systems. In *Proc. DSN*, 2012.

[KGN08] S. Kavulya, R. Gandhi, and P. Narasimhan. Gumshoe: Diagnosing performance problems in replicated file-systems. In *Proc. SRDS*, 2008.

[KTGN10] M. P. Kasick, J. Tan, R. Gandhi, and P. Narasimhan. Black-box problem diagnosis in parallel file systems. In *Proc. FAST*, 2010.

[LFY+10] 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.

[McD89] C. McDiarmid. On the method of bounded differences. *Surveys in Combinatorics*, 1989.

[MRS08] C. D. Manning, P. Raghavan, and H. Schüze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[MSA08] V. Mnih, C. Szepesvári, and J.-Y. Audibert. Empirical bernstein stopping. In *Proc. ICML*, 2008.

[NDO11] 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.

[NM07] A. B. Nagarajan and F. Mueller. Proactive fault tolerance for HPC with xen virtualization. In *Proc. ICS*, 2007.

[OAS08] A. J. Oliner, A. Aiken, and J. Stearley. Alert detection in system logs. In *Proc. ICDM*, 2008.

[PBYH+08] 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.

[PLSW06] N. Palatin, A. Leizarowitz, A. Schuster, and R. Wolff. Mining for misconfigured machines in grid systems. In *Proc. SIGKDD*, 2006.

[Ran89] R. H. Randles. A distribution-free multivariate sign test based on interdirections. *Journal of the American Statistical Association*, 1989.

[SM09] R. Serfling and S. Mazumder. Exponential probability inequality and convergence results for the median absolute deviation and its modifications. *Statistics & Probability Letters*, 2009.

[SOR+03] 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.

[Tuk75] J. Tukey. Mathematics and picturing data. In *Proc. ICM*, 1975.

[XHF+09] 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.

[ZCG+05] 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.

אמיתית על ידי מערכת הניטור הקיימת.

סיכום

הגישה הרווחת לניטור מרכזי מחשוב מתרכזת בזיהוי תקלות שכבר קרו והתאוששות מהירה. עבודה זו מציעה אלטרנטיבה: אנו מראים שכשלים חבויים הם נפוצים, וניתן לזהות אותם זמן רב לפני שהם הופכים לתקלה במחשב.

אנו מציעים מגנון פרקטי לזיהוי כשלים חבויים. המגנון המוצע ניתן לשימוש במגוון סוגי מערכות, תומך באלפי מחשבים, ועמיד לשינויים לאורך זמן. הוא הצליח לזהות תקלות ימים רבים מראש ובדיוק גבוה. המנגנון מספק חסם סטטיסטי על קצב התראות השווא, שהוכח בניסויים על מספר סוגי מערכות. הגישה שלנו גמישה וקלה לשימוש: אין צורך באימון המנגנון או כיוונון פרמטרים, ואין צורך במומחיות ובתיוג רשומות הסטוריות.

כל הזמנים כך שכל מחשב מקבל ציון חריגה, אשר מתאר עד כמה הוא שונה בממוצע מהמחשבים האחרים.לבסוף, עבור פונקציות בדיקה המצייתות למספר כללים פשוטים על טווחי הציונים שהן יכולות להעניק, אנו מספקים חסמים סטטיסטיים אשר עונים על השאלה הבאה: בהינתן ציון החריגה של מחשב, מה ההסתברות שמחשב תקין יקבל ציון חריג כל כך? אם הסתברות זו נמוכה מדי, מכריזים על המחשב כחשוד.

בעבודה זו אנו מציגים ובוחנים שלוש פונקציות בדיקה שונות במסגרת המנגנון הגנרי, כולן מסתכלות על מדדי הביצועים של המחשבים כנקודות במרחב רב מימדי. הראשונה בודקת את הכיוון הממוצע ממדדי המחשב לאלה של שאר המחשבים. במחשב תקין הכיוונים יהיו אקראיים, בעוד שאם יש כשל חבוי אז חלק מהמדדים של המחשב יהיו תמיד נמוכים או גבוהים יחסית לאחרים. פונקצית הבדיקה השניה מניחה שמדדי הביצועים יוצרים מעין ענן של נקודות, והיא בודקת עד כמה נקודה היא פנימית בתוך הענן. אנומליות ביצועים תהיינה בדרך כלל מחוץ לענן. פונקצית הבדיקה השלישית משווה את הצפיפות המקומית של הנקודות המתארות מחשבים שונים. המדדים של מחשבים חריגים יהיו לרוב באזור יחסית דליל, שכן רוב המחשבים לא תקולים ולכן לא דומים למחשב החריג.

ניסויים

בחנו את ביצועי מנגנון זיהוי הכשלים החבויים על מספר מערכות אמיתיות. המערכת העיקרית שבה השתמשנו היא חלק ממנוע חיפוש גדול הנמצא בשימוש נרחב, והיא מורכבת מכ־4500 מחשבים. בנוסף, מאחר שמדובר במרכז מחשוב אמיתי וחי, השתמשנו בתשתיות הקיימות שלו אשר אוספות מדדי ביצועים וכן מנהלות רישום של תקלות ממערכת הניטור הקיימת (אשר מבוססת על חוקים).

עקבנו אחרי ביצועי המערכת הגדולה במשך 60 יום – בכל יום הרצנו בדיקה לזיהוי כשלים חבויים, ואז השווינו את התוצאות לרישומי מערכת הניטור הקיימת במשך מספר ימים לאחר מכן. כאשר אנו מזהים כשל חבוי במחשב כלשהו, יש סיכוי של כ־70% שמחשב זה יסבול מתקלה במהלך 15 הימים הקרובים, וזאת לעומת 20% סיכוי במחשב אקראי כלשהו. קצב התראות השווא (false positive rate) היה כ־2%, קרוב לתכנון המקורי (1%). בנוסף, מנגנון זיהוי הכשלים הראה את גמישותו כאשר המערכת המנוטרת עברה עדכוני גרסה חיים בזמן הניטור. לאחר סיום העדכון, מנגנוני הניטור חזרו מיד לביצועים טובים ללא צורך בהתאמה או כיוונון פרמטרים.

בניסוי נוסף, סרקנו רשומות היסטוריות יום אחד לפני תקלות ידועות במערכת הגדולה, וכן עבור מחשבים ללא תקלה. עבור פרמטר שמרני של 1% התראות שווא, נמצאו כשלים חבויים בכ־20% מהמחשבים, וללא שום התראת שווא במחשבים התקינים.

כמו כן בחנו את הביצועים במשך 60 יום על שתי מערכות קטנות יותר אשר אינן מקיימות את ההנחות שלנו לגבי איזון עומסים: מערכת שמירת ערכים (key-value store) בעלת כ־300 מחשבים ומערכת הגיבוי שלה (גם 300 מחשבים). למרות שהן לא מקיימות את ההנחות שלנו, ולמרות שרישומי התקלות שהיו לנו היו חלקיים, ההבטחות הסטטיסטיות שלנו התקיימו – קצב התראות השווא היה כ־2%, אולם הדיוק בזיהוי תקלות האמת היה נמוך.

לבסוף, בחנו את מנגנון זיהוי הכשלים החבויים על מערכת קטנה שרצה על 15 מחשבים וירטואלים במשך 30 יום. בזמן זה זוהו 2 כשלים חבויים בלבד, ואחד מהם זוהה מספר ימים אחר מכן כתקלה

שיטות למערכות ספציפיות

עבור מערכות ספציפיות, לדוגמא סוגים מסויימים של מערכות קבצים מבוזרות, קיימות שיטות גמישות אשר לא לומדות מהעבר. שיטות אלה משתמשות בתובנות עמוקות לגבי המערכת הללו, כגון בחירת מדדי ביצועים, פרמטרים מתאימים, וספים לחיזוי. הן מותאמות לסוג מסויים של מערכות, ואילו מטרת עבודה זו היא למצוא שיטה שתעבוד על סוגים רבים של מערכות מתחומים שונים.

עבודה זו: זיהוי מוקדם של תקלות ואנומליות

גישה חדשה לבעית הניטור מתרכזת בזיהוי מוקדם של בעיות ביצועים, או כשלים חבויים. **כשל חבוי** הוא התנהגות של מחשב שמצביעה על תקלה, או יכולה להתבטא כתקלה לאחר זמן, אולם היא "טסה מתחת לרדאר" של מערכות ניטור כיוון שהיא לא בולטת מספיק, או כיוון שתקלה כזו לא נצפתה מראש על ידי מתכנני מערכת הניטור. האתגר בתכנון מנגנון גנרי לזיהוי כשלים חבויים הוא שהמנגנון צריך להיות גמיש דיו על מנת להתמודד עם שינויים במערכת ובין מערכות שונות. הוא צריך לזהות כמה שיותר כשלים אך למזער את התראות השווא.

עבודה זו מתארת שיטה סטטיסית גנרית ללא פיקוח (unsupervised) לזיהוי כשלים חבויים. השיטה המתוארת לא דורשת ידע על המערכת המנוטרת, ולא דורשת שינויים חודרניים במערכת. היא משתמשת אך ורק במדדי הביצועים אשר נאספים מכל מחשב במרכזי מחשוב גדולים באופן שגרתי. הניסויים שלנו מראים שכשלים חבויים קדמו זמן רב ללפחות 20% מהתקלות במערכת אמיתית עם מעל 4500 מחשבים, ואשר זוהו ללא התראות שווא. בנוסף, הראינו שבעזרת גילוי כשלים חבויים ניתן לחזות תקלות במחשבים עד שבועיים מראש ברמת דיוק (precision) של עד 70%.

השיטה: זיהוי כשלים על ידי מציאת אנומליות

שירותי רשת מקוונים גדולים צריכים להתמודד עם עומס רב, ועל כן הן בנויים באופן סקלבילי (scalable). אחת הדרכים הנפוצות היא שכפול – השירות משוכפל על הרבה מחשבים זהים (או על מספר קטן של קונפיגורציות), וקיים מנגנון של איזון עומסים (load balancing) אשר מחלק את העבודה בין המחשבים.

התובנה הבסיסית היא שבמערכת כזו שבה כל המחשבים עושים אותו דבר, רוב המחשבים תקינים רוב הזמן, וכיוון שהם זהים, אנחנו מצפים שמדדי הביצועים של המחשבים ישקפו התנהגות דומה בממוצע. על כן הרעיון המרכזי בשיטה היא להשוות את מדדי הביצועים של המחשבים במספר נקודות זמן, ואז למצע את תוצאות ההשוואות לאורך מספר זמנים שונים.

השיטה מורכבת מארבע שלבים: ראשית, מסננים את מדדי הביצועים ומנרמלים אותם על מנת להפטר ממדדים שאינם מועילים. תהליך זה אוטומטי לחלוטין וניתן לעשות אותו מראש. בשלב שני, בכל נקודת זמן, משווים את מדדי כל המחשבים זה לזה בעזרת פונקצית בדיקה כלשהיא. פונקציה זו מקבלת את מדדי כל המחשבים בזמן מסויים, משווה אותם, ומעניקה לכל מחשב ציון אשר מתאר עד כמה הוא שונה מהאחרים בנקודת זמן זו. לאחר מכן ממצעים את הציון של כל מחשב לאורך

במערכת המנוטרת.

ניטור מבוסס חוקים

באופן מעשי, רוב הניטור במערכות קיימות מבוסס על הגדרת סט של חוקים וניטורם. רוב החוקים מוגדרים על מדד ביצועים בודד של מחשב אחד או השירות כולו, לדוגמא טמפרטורת המעבד או המקום הפנוי בדיסק. כאשר המדד חוצה סף שהוגדר מראש, מערכת הניטור תבצע פעולה כגון הודעה למפעילי המערכת, או התאוששות אוטומטית על ידי אתחול מחדש.

ניטור על ידי חוקים הוא בעייתי מכמה סיבות. מצד אחד יש לקבוע ספים הדוקים מספיק על מנת לאתר תקלות. מצד שני, אסור לקבוע ספים הדוקים מדי כדי למנוע התראות שווא. אולם כיוון שעומס העבודה משתנה כל הזמן וקשה לחזותו מראש, שום סף קבוע אינו מספיק. בנוסף, למערכות שונות, ואפילו לגרסאות שונות של אותה מערכת, יש נקודות עבודה שונות. עקב כך החוקים צריכים תחזוקה ידנית מתמדת, אשר לרוב נעשית מאוחר מדי, כתוצאה מתחקיר על תקלה שלא זוהתה, או התראת שווא יקרה.

לימוד מהעבר

גישות מתקדמות יותר ממדלות את פעולת המערכת בעזרת מידע היסטורי, לרוב בעזרת למידה עם פיקוח (supervised learning). בהנתן רשומות הסטוריות של המדדים, ובנוסף תיוגים המתארים אם יש או אין תקלה במערכת באותו זמן, ניתן ללמוד מודל אשר ינטר את המדדים ויזהה תקלות חדשות. לגישות כאלה קשה להסתגל לשינויים התכופים בעומס העבודה ובמערכת המנוטרת. לאחר שינויים כאלה ההיסטוריה והמודל שנבנה כבר אינם רלוונטים. בעייה נוספת היא הצורך במידע הסטורי מתוייג, והצורך לתייג אותו מחדש לאחר שינויים, תהליך שעלול להיות מסורבל ויקר.

ניתוח פלט טקסטואלי

שיטות גמישות יותר קיימות בתחום של מחשוב עתיר ביצועים (high performance computing). שיטות טיפוסיות מנתחות הופעות של שורות בפלט הטקסטואלי של התוכנה על מנת לזהות תקלות במערכת או במחשבים ספציפיים. שיטות אלה אינן מעשיות עבור שירותים מקוונים עתירי תעבורה מסיבות של ביצועים ורוחב פס. שרותים מקוונים גדולים מבצעים הרבה מאוד פעולות קצרות, בקצב גבוה ובמהירות רבה. במקרה זה פלט טקסטואלי לא קיים (או מוגבל ביותר), ובמקום זה המחשבים סוכמים מדדי ביצועים ומדווחים באופן מחזורי. דיווחים אלה זהים בקצבם ובהופעתם גם עבור מחשבים איטיים או תקולים. הערכים המדווחים עצמם הם החשובים, ולא ההופעה של הדיווח עליהם. כמו כן, פלט טקסטואלי מקורו בתוכנה הרצה ולכן לרוב ישקף תקלות תוכנה. אנו מעוניינים במדדי ביצועים מכל שכבות המערכת על מנת לזהות גם בעיות חומרה וגם בעיות תוכנה.

תקציר

בשנים האחרונות הדרישה לכוח חישוב ואחסון מידע גדלה. שירותי רשת מודרניים וחישוב בענן (cloud computing) מסתמכים על מרכזי מחשוב גדולים אשר לרוב בנויים מאלפי מחשבים. במערכות כה גדולות אי אפשר להניח שכל המחשבים תקינים ומוגדרים כראוי.

ניטור המחשבים הללו הוא חיוני במרכזי מחשוב, כיוון שתקלות שלא שמו לב אליהן עלולות להצטבר ולגדול עד לנקודה שבה יכולות היתירות של המערכת ומנגנוני האל־כשל שלה נכשלים. ניטור ידני הוא לא מעשי במרכזי מחשוב עקב המספר הגדול של המחשבים. במקום ניטור ידני, המחשבים מנוטרים באופן אוטומטי על ידי איסוף מספר גדול של מדדי ביצועים וניתוחם. כל מחשב מדווח מאות מדדי ביצועים שמקורם בכל שכבות המערכת, החל ממדדים של השירות עצמו (כגון מספר השאילתות בתור) ועד מדדים כלליים (כגון ניצול המעבד).

עבודה זו מתארת מתארת מנגנון גנרי לניטור וזיהוי כשלים חבויים – אנומליות ביצועים אשר מעידות על תקלה קיימת במחשב, או יכולות לגרום לתקלה בעתיד. מנגנון זה לא צריך מידע הסטורי וגם לא ידע ספציפי על המערכת המנוטרת. המנגנון גמיש ומתמודד עם שינויים בעומס העבודה ובמערכת המנוטרת, וכן מספק חסמים סטטיסטיים על קצב התראות השווא (false positive rate).

הניסויים שביצענו מספקים עדויות לכך שתקלות חבויות נפוצות אפילו במרכזי מחשוב המנוהלים בקפידה, ושניתן לזהות אותם זמן רב מראש ובדיוק גבוה, וזאת ללא צורך בתובנות על המערכת המנוטרת, בלמידה מהעבר, או בהתאמת המנגנון למערכת הספציפית.

רקע: ניטור מחשבים לזיהוי תקלות

האתגר בניטור שירותי רשת מקוונים (online) גדולים נובע מהסביבה הדינמית והלא צפוייה שבה הם נמצאים. ראשית, לרוב קשה לחזות מראש את עומס העבודה – הוא בלתי צפוי ומשתנה כל הזמן. שנית, עדכונים תכופים של התוכנה (ולעיתים אף החומרה) משנים את ההתנהגות הבסיסית הטיפוסית של המערכת. שלישית, קשה ויקר להשיג הסטוריה מבוארת של התנהגות המערכת, שכן הדבר דורש ממומחים בעלי ידע עמוק על פעולת המערכת לעבור באופן ידני על הרשומות ההיסטוריות של המדדים ולתייגם. ולבסוף, התראות שווא עלולות להיות יקרות שכן הן מערבות מהנדסי תמיכה אשר צריכים להגיב להתראה. בנוסף, התראות שווא רבות עלולות לגרום לתופעה של התעלמות מהתראות אמת, שנגרמת כאשר כוח האדם המטפל בהתראות מתחיל להתעלם מהן כי רובן התראות שווא.

שיטות הניטור הקיימות סובלות ברובן מבעיה של חוסר גמישות – הן לא מסתגלות לשינויים התכופים

המחקר בוצע בהנחייתם של פרופסור אסף שוסטר בפקולטה למדעי המחשב ודוקטור רן גלעד־בכרך מחברת מיקרוסופט.

חלק מן התוצאות בחיבור זה פורסמו במהלך תקופת המחקר של המחבר כמאמר בכנס מאת המחבר ושותפיו למחקר, אשר גרסתו העדכנית ביותר היא:

M. Gabel, A. Schuster, R.-G. Bachrach, and N. Bjorner. Latent fault detection in large scale services. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–12, 2012.

תודות

ראשית, אני רוצה להודות ליועצים שלי, פרופסור אסף שוסטר ודוקטור רן גלעד־בכרך, על ההנחייה והסבלנות הרבה. למדתי רבות מהם, והיכן שלא - האשמה היא בי. ההכוונה שלהם והעידוד המתמיד לאורך השנים הם אלה שאפשרו את קיומה של עבודה זו.

אני רוצה גם להודות למחבר השותף שלנו, ניקולאי ביורנר, על הצעותיו הנבונות ועל חוש ההומור שלו בזמנים לחוצים.

לכל חברי, שנתנו לי להתאמן עליהם בהעברת הרצאות, ואף דברים גרועים יותר - תודה! אתם יודעים מי אתם.

לסיום, תודה מיוחדת למשפחתי. לא הייתי מגיע לקו הסיום ללא תמיכתם התמידית.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

זיהוי אנומליות ללא פיקוח
במרכזי מידע גדולים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במדעי המחשב

משה גבל

הוגש לסנט הטכניון – מכון טכנולוגי לישראל
סיון התשע"ג חיפה יוני 2013

זיהוי אנומליות ללא פיקוח
במרכזי מידע גדולים

משה גבל