IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 May 2023, accepted 13 June 2023, date of publication 10 July 2023, date of current version 17 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3293643

RESEARCH ARTICLE

Detecting Anomalies Through Sequential Performance Analysis in Virtualized Environments

CHARLES F. GONÇALVES[1,4], DANIEL SADOC MENASCHÉ[2], (Member, IEEE), ALBERTO AVRITZER[3], (Member, IEEE), NUNO ANTUNES[1], (Member, IEEE), AND MARCO VIEIRA[1], (Member, IEEE)

[1]Centre for Informatics and Systems of the University of Coimbra (CISUC), Department of Informatics Engineering, University of Coimbra, 3030-290 Coimbra, Portugal
[2]Federal University of Rio de Janeiro, Rio de Janeiro 21941-901, Brazil
[3]EsulabSolutions Inc., Princeton, NJ 08536, USA
[4]Federal Center for Technological Education of Minas Gerais (CEFET-MG), Board of Information Technology, Belo Horizonte 30421-169, Brazil

Corresponding author: Charles F. Gonçalves (charles@dei.uc.pt)

This work is funded by Project "Agenda Mobilizadora Sines Nexus". ref. No. 7113), supported by the Recovery and Resilience Plan (PRR) and by the European Funds Next Generation EU, following Notice No. 02/C05-i01/2022, Component 5-Capitalization and Business Innovation-Mobilizing Agendas for Business Innovation, by national funds through the FCT-Foundation for Science and Technology, I.P., within the scope of the project CISUC-UID/CEC/00326/2020, grant SFRH/BD/144839/2019, by European Social Fund, through the Regional Operational Program Centro 2020, and by CEFET-MG and partially by CAPES, CNPq, and FAPERJ under grants 315110/2020-1, E-26/211.144/2019 and E-26/201.376/2021.

ABSTRACT Virtualization enables cloud computing, allowing for server consolidation with cost reduction. It also introduces new challenges in terms of security and isolation, which are deterrents for the adoption of virtualization in critical systems. Virtualized systems tend to be very complex, and multi-tenancy is the norm, as the hypervisor manages the resources shared among virtual machines. This paper proposes a methodology that uses performance modeling for the detection of anomalies in virtualized environments that can be caused, for instance, by cyberattacks. Experiments are conducted to profile the system operation under normal conditions for its business transactions. The results are used to calibrate a performance model and to understand the impact of its parameters on the false positive probability. During operation, the system is monitored, and deviations are detected by applying a sequential analysis algorithm (the bucket algorithm). The methodology is evaluated using a representative cloud workload (TPCx-V), which was profiled during a set of controlled executions. We consider resource exhaustion anomalies to emulate the effects of attacks affecting the performance of the system. Our results show that the proposed approach is able to successfully detect anomalies, with a low number of false positives, and spot possible residual effects of anomalies on the system.

INDEX TERMS Anomaly detection, modeling, performance, security, virtualization.

I. INTRODUCTION

Virtualization environments enable the multiplexing of physical resources among many Virtual Machines (VMs) [1]. In a virtualized environment, cloud providers that use an Infrastructure as a Service (IaaS) model to share their infrastructure with multiple tenants rely on a virtual machine manager, also known as hypervisor [2], to allocate physical resources to each virtual machine. Efficient, fair, and secure resource allocation are critical aspects of the IaaS model.

The ubiquity of cloud solutions [3] is a potentially fertile ground for security and privacy breaches [4], [5], [6], [7]. **In a multi-tenant environment, a legally acquired virtual host may initiate security attacks**, where a malicious user exploits hypervisor security vulnerabilities to attack a tenant that shares the same physical resources. Detecting and mitigating such attacks is an important step to counter the threat posed to the existing IaaS systems and, more broadly, to the virtualization culture [8].

The associate editor coordinating the review of this manuscript and approving it for publication was Vlad Diaconita.

Some systems only use virtualization capabilities to simplify the management of standalone applications, benefiting from some functionalities that come with cloud computing. But there are other systems that spread across multiple servers, splitting their services into different components. These orchestrated components, potentially located in distinct physical servers providing multiple different types of services (e.g., transactional systems, interface handling, long batching jobs, etc), compose a complex system and work together to support the business models and operations. In this work, we refer to such configurations as complex virtual systems.

The design of intrusion detection systems (IDSs) for detecting anomalies, such as zero-day attacks and advanced persistent threats (APTs) [9], [10] in virtualized environments, raises several domain-specific challenges [11], [12]. In particular, (*i*) it is challenging to comprehensively define normal behavior in a diverse cloud environment, (*ii*) malicious attackers may adapt their behavior to fit the domain definition of ''normal behavior'', and (*iii*) data on anomalies at cloud environments, which would be instrumental for training purposes, are hard to obtain. To tackle those challenges, we focus on anomaly detection approaches based on system performance signatures. In particular, performance signatures have the ability to address any attack that impacts the overall system performance, including the potential of detecting zero-day attacks [11], [12], as those approaches are based on detecting performance deviations and do not require detailed knowledge of attack history [12].

In this context, we pose the following research question:

> **RQ:** *Is it feasible to* ***efficiently tune*** *mechanisms for anomaly detection in complex virtualized systems* ***trading off between contending factors*** *such as the time to detect anomalies and the rate of false-positives under a principled model-based framework?*

This paper **proposes a methodology for anomaly detection in complex virtualized systems based on performance deviations**. Initially, the methodology profiles the system operation *under normal conditions* by computing the mean and standard deviation throughput of every transaction in the target system, establishing its baseline profile. Then, during system operation, performance is monitored to capture deviations from the baseline profile, using the bucket algorithm [13] to signal the anomalies following the tuning strategy. **The proposed tuning of the anomaly detection mechanism leverages a calibrated analytical model that is used to control the rate of false-positives in a principled manner [11]**.

The methodology was validated by running the TPCx-V [14] workload, which was designed to model large virtualized infrastructures that support transactional systems. Security attacks that impact system performance are emulated by using fault injection (related work on dependability assessment based on fault injection can be found in [15] and [16]).

The experiments show the applicability and effectiveness of the proposed anomaly detection methodology. In fact, in our experiments, it was possible to detect most of the performance deviations, with a low number of false-positives (precision of 90% and 98% for the worst and best configurations). The methodology is also instrumental in identifying the residual effects of failures: we observed more prominent transient residual effects after anomalies that caused more significant performance degradation during their active period. Indeed, severe failures imply extended recovery periods [17], [18].

Our proposal has the **advantage of being less intrusive than alternatives that rely on deep packet inspection for monitoring and anomaly detection [6], [19]**. In fact, it is only necessary to monitor the throughput of the business transactions, making it less dependent on the supporting technology stack and, therefore, more portable. Another advantage is the anomaly detection algorithm simplicity, which is easier to train, understand, and tweak than algorithms that need complex learning processes. These factors make the overall methodology easier to use and more applicable in practice.

In summary, our contributions are the following:

(*i*) A novel **anomaly detection methodology** that monitors business transactions throughput using the bucket algorithm (Section III).

(*ii*) An **analytical model that can be used to parameterize and control the anomaly detection mechanism**. The model can be used to manage the tradeoff between the time to detect an attack and the rate of false alarms (Section IV).

(*iii*) An **experimental assessment of the methodology in practice** using a representative system and attacks. We established the feasibility of detecting attacks based on non-intrusive user-level performance metrics that are available in production environments (Sections V and VI).

(*iv*) A **model-driven principled mechanism design** that supports what-if assessment of the anomaly detection algorithms parameterization (Section V-C).

This paper is an extended version of [20], where the anomaly detection approach with its analytical performance modeling was presented and validated. Among the novelties in the extension, we present 1) an attack detection methodology encompassing exploratory, profiling, and operation phases (Section III), 2) a unified framework for sequential analysis, having the bucket algorithm as a special case (Section IV-F), 3) an additional case study (Section VI), and, 4) derivation of formal results in Appendices A to C.

The remainder of the paper is organized as indicated in the summary of contributions above. Section II covers related work, followed by our contributions in Sections III-VI and Section VIII concludes and discusses future work.

II. RELATED WORK

This section reviews prior art and compares it with the research presented in this paper. There is a vast literature on machine learning techniques for anomaly detection, applying

Convolutional Neural Networks [21], Long Short Term-Memory [22] and Sequential Deep Learning [23]. There are also works using classical techniques such as CUSUM [24], [25], [26]. Those techniques are usually applied over the underlying system performance metrics (CPU, memory, etc.) to detect operational anomalies. In our work, we are interested in assessing the use of sequential testing techniques over the service throughput metric to evaluate the capability of detecting malicious anomalies under a virtualized system.

A. SECURITY ISSUES RELATED TO CLOUD MULTI-TENANCY

Multi-tenancy with isolation among tenants is the most widely used model in cloud-computing environments. However, regardless of the hardware isolation mechanisms used, there have been many recently reported security intrusions across this isolation barriers [27]. In [28] and [29], the authors discuss security issues related to running the hypervisor in cloud environments. For example, a malicious virtual machine running over the hypervisor could attempt to break the hardware isolation barrier, violate privacy rules, or attempt to compromise the hypervisor, e.g., through a system resource exhaustion causing a Denial-of-Service (DoS) attack. Those works also discuss architecture features that are required to mitigate security vulnerabilities. In this paper, we focus on detecting anomalies in operational systems under the assumption that the aforementioned vulnerabilities cannot be fully prevented.

The tracking of system calls can be used to implement IDSs for virtualized environments, as shown in [30]. The idea consists of watching for, during execution, the anomalous system call sequences in the user programs. The assessment in [30] used a Linux KVM virtual machine to collect the system calls from user programs. Results show a tradeoff between detection and time efficiency when considering different system call sequence lengths. Nonetheless, benchmarking the performance of IDS tools is a challenging problem [31], motivating simpler and interpretable detection methods. Our work is complementary to [30], as we indicate that IDSs can benefit from anomaly detection approaches that are less intrusive than those reported in [30], relying on [30] only on specific scenarios.

B. ANOMALY DETECTION FOR CYBERSECURITY

An approach for anomaly detection consists in running sequential hypothesis tests [32], [33], [34]. In [34], sequential hypothesis tests are used for the detection of malicious port scanners. The authors have developed a link between the detection of malicious port scans and the theory of sequential hypothesis testing. They have also shown that port scanning can be modeled as a random walk. The detection algorithm matches the random walk to one of two stochastic processes, which correspond to malicious remote hosts or authorized remote hosts. The approach considered in our paper is similar in spirit to that considered in [33] and [34], as our analytical results are derived from a birth-death Markov chain. This Markov chain can be interpreted as a random walk, where the system of buckets will fill as the system degrades and empty as the system recovers (see Section IV-C).

Previous works have considered anomaly detection approaches using performance signatures [35], [36], [37], [38]. The work in [36] introduces a framework that detects anomalous application behavior using regression-based models and application performance signatures. This method uses the concept of profiling system resources, such as CPU demand. Then, [37] builds on top of [36] and proposes an anomaly detection approach based on performance signatures from CPU, I/O, memory, and network usage for the detection of security intrusions. The approach introduced in this paper outperforms IDSs based on security intrusion history [12] because the latter cannot detect zero-day attacks. The approach on [38] uses Principal Component Analysis (PCA) over the system metrics to reduce the number of evaluated metrics and uses a reliability model to dynamically monitor the system for anomalies increasing the timeliness of the anomaly detection.

Another technique to detect anomalous behavior is chaos engineering [39], [40]. The main idea is to hypothesize a steady-state by a measurable output, and create ''chaos'' into the system, i.e., inject some faults to cause errors. Afterward, evaluate whether the hypothesis that the system will behave properly holds. Despite not being explicitly designed for security, previous work [41] has applied chaos engineering to evaluate the security of public clouds. The methodology of our work can be applied under the framework of chaos engineering to identify anomalies, especially in virtualized systems.

C. BUCKET ALGORITHM AND SEQUENTIAL DECISION MAKING

The performance of signature-based intrusion detection systems relies on intrusion detection algorithms that account for workload variability to avoid a high rate of false-positive alerts. The Bucket Algorithm (*BA*), introduced in [42], is an example of a workload-sensitive algorithm. Anomaly detection using the *BA* inspects the last sample of the metric of interest and compares it against the calibrated mean and standard deviation of the same metric. Then, it relies on the central limit theorem [13] to assess if the last samples of the metric comply with normal behavior or represent an anomaly. In Section IV we revisit the *BA* mechanism and present an analytical model that is instrumental to parameterizing the *BA* from experimental data.

Sequential probability ratio test [33], [34] and the *BA* mechanism have been previously proposed to detect attacks in networks. Their analysis involves a random walk birth-death process. According to the bucket algorithm, for instance, in the initial state, all buckets are empty, and in the final state, all buckets are full. Then, a birth-death process captures the random walk corresponding to balls being added

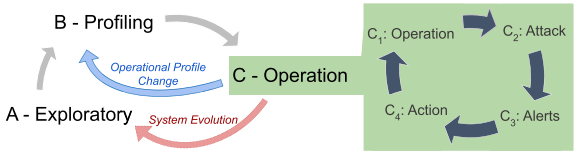


FIGURE 1. Overview of the methodology application life cycle.

and removed from the buckets. Understanding the mean time to reach the final state, one can tune the number of buckets and the depth of each bucket in a principled way (see Section IV).

D. COVERT ATTACKS

Our approach cannot detect attacks that are purposely designed to be covert. For instance, it is known that there are theoretical limits for the detectability of volume-based attacks, including certain sorts of DDoS attacks [32], [43]. Despite the fact that those attacks are theoretically guaranteed to be covert when the number of bytes transmitted through the network is the sole measure available for the defender, the joint measurement of memory and CPU usage may allow the detection. The joint analysis of multiple metrics for detecting attacks that are purposely designed to be covert is beyond our scope and is left as a subject for future work.

III. ATTACK DETECTION METHODOLOGY

The attack detection methodology introduced in this paper supports the anomaly detection approach by coping with changes in the operational profile, as the monitored systems commonly have different loads and demands over time.

Figure 1 depicts the high-level application of the methodology. As we can observe, it consists of three main phases: **(A) exploratory analysis**, which allows determining the most effective way to monitor the system based on its characteristics; **(B) profiling**, which evaluates the system on its expected regular operational profile(s) [44] to extract information about the habitual behavior and its performance; and **(C) operation**, in which the system executes for its intended purposes, and we use all data and knowledge from the previous phases to report deviations, identifying them as anomalies or attacks.

The utilization of cloud computing environments may result in a variation in the accessibility of provided resources such as CPU, RAM, and disk I/O [45]. In extreme cases, a VM or a group of VMs that share the same physical hardware consume an excessive amount of resources (such as CPU, memory, or disk I/O), affecting the performance of other VMs sharing the same host. Such ''noisy neighbors'' [45], [46] cause anomalous effects in the environment. To avoid such excessively noisy events, numerous approaches could be employed, such as obtaining reserved capacity and procuring service tiers with stronger isolation guarantees, which are typically available in most cloud providers. In this study, we assume that the system under test might be vulnerable to noisy neighbors, and as a result, it is critical for the system maintainers to be aware of such occurrences.

It is known that the throughput of a single workload can vary over time [47], and the methodology handles transient variations in throughput. Still, if the workload changes over a coarser time scale (e.g., holiday/promotion seasons), the operational profile will change, and an iteration to accommodate that new profile is needed, noting that some profiles are expected to repeat from time to time (e.g., during holiday seasons). Also, any changes in the system may require an evaluation of relevant aspects to add to the monitoring surface. The optimal integration of a change detection mechanism into the overall monitoring system is out of the scope of this paper and is left as a subject for future work.

When considering security issues related to virtualization [48] and associated attacks [49], [50], our approach can be applied to various types of anomalies and attacks that arise in complex systems. Our approach can detect attacks that cause a system component to halt and disrupt the overall throughput. Examples of such attacks that compromise the running state of the hypervisor and its VMs include Hypervisor/VM Crash and Resource Starvation [49]. Additionally, our approach can also be applied to other categories of attacks that impact the hypervisor and its VMs, degrading the overall throughput at varying intensities, such as classical network attacks, VM sprawl, and resource exhaustion. As these attacks alter the throughput distribution, our approach is a candidate to detect such anomalies. In Section V, we illustrate the applicability of our approach using a resource exhaustion attack.

The attack classification presented by [48] establishes a hierarchical structure with multiple attack classes, where Compromise-Based Attacks are at one of the top levels. Such attacks have the potential to compromise any component of a complex system, leading to situations that disrupt the overall system throughput. In the event of such situations, our approach can be used to detect anomalies and assist system maintainers in addressing those anomalies.

The phases of the proposed methodology are detailed in Figure 2, and explained in the ensuing sections.

A. EXPLORATORY ANALYSIS PHASE

This phase has the objective of determining the surface used to monitor the system performance. To achieve that goal, the user needs to follow three main steps:

A.1) *Analysis:* evaluate the internal aspects of the system, such as the architecture, components, operations, and resources that are available for monitoring to define how to characterize the system performance.

A.2) *Exploration:* the tacit knowledge of the system is essential, and a set of executions (**E** on Figure 2) is prescribed for exploratory analysis. During those executions, all monitored data should be collected and analyzed to understand the metrics that effectively translate into useful measurements.

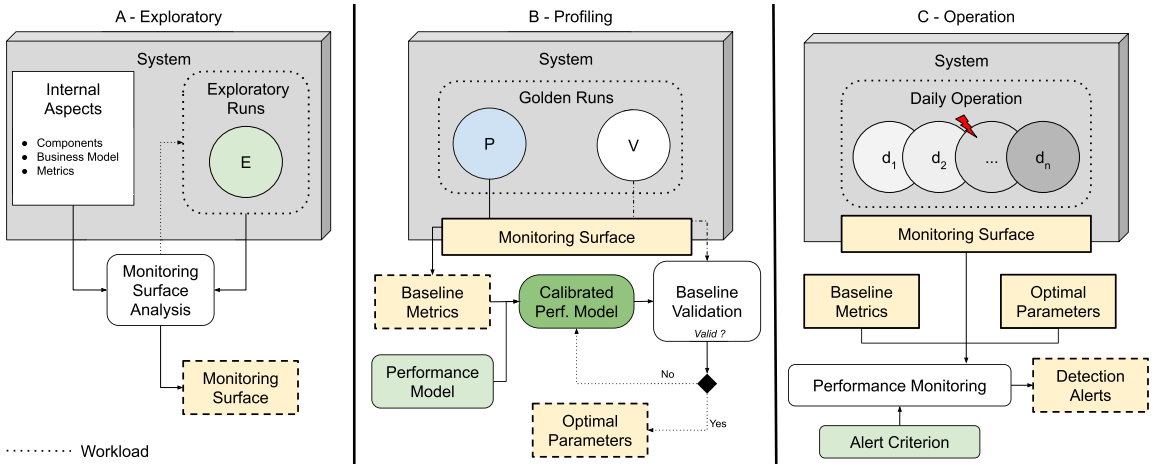


FIGURE 2. Diagrams showing the three distinct sets of runs present on our methodological approach: Exploratory, Profiling, and Operation.

A.3) *Definition:* use the outputs of previous steps to determine the **monitoring surface**, i.e., combine the tacit knowledge and the internal aspects to define which components and metrics to use in the following phases.

B. PROFILING PHASE

This phase has the objective of evaluating the environment during the execution of its expected regular operational profiles [44] to establish the statistical parameters that represent those profiles. Note that by conducting profiling within a cloud provider, our approach intrinsically captures a number of effects in the regular operational profiles, such as the tolerated performance interference across VMs that is often observed in cloud environments, as noted in [46]. The following procedures are needed:

B.1) *Golden runs definition:* collect data under normal operating conditions using the monitoring surface. These **golden runs** are the primary reference for system behavior without faults or attacks. We split these runs into two independent sets: a first one, **P**rofile, used to compute the baseline metrics, and a second one, used to **V**alidate the performance parameterization.

B.2) *Baseline metrics extraction over the monitoring surface:* compute the baseline metric for each subsystem, operation, resource, and profile. Extract statistical data from the metrics defined over the monitoring surface.

B.3) *Performance model calibration:* apply the empirical data to determine the optimal values that minimize the number of false-positives (FP) alerts. We are assessing experimental runs under no attack conditions. The performance model determines which parameters are best suitable for the tests. Then, we account for the number of alerts that a given setting will cause. Therefore, we can define the algorithm tolerance for the number of FPs according to user preferences. The performance model will assist in minimizing the FP rate.

B.4) *Baseline validation:* check the validity of the determined parameters by applying the detection algorithm using the Golden Runs from the **V** group and check if the number of alerts (FP) is in the defined criteria.

B.5) *Parameterization adjustment:* during the baseline validation, calibrate the model until the alert rate meets the defined objective.

C. OPERATION PHASE

At this point, the anomaly detection algorithm is fully configured to monitor the system in production and detect security attacks that cause performance degradation. Thus, in this phase, we take advantage of all the data and knowledge from the previous ones to search for anomalies or attacks during the system's operation. We need to:

C.1) *Define a criterion:* that concisely determines an alert condition. As false-positives can be expected in some transactions, we need to determine if an alarm in a specific transaction will trigger an alert or not. For instance, only alarms in critical subsystems, or multiple alarms in certain transaction subsets, will trigger an alert.

C.2) *Monitor the operation:* integrate the detection algorithm into the daily activity by continually watching the monitoring surface using the detection algorithm with the optimal parameters.

Finally, the system's administrators must take into consideration the need to monitor the change in the operational profile, which will trigger a new iteration of the methodology

IV. ANOMALY DETECTION MECHANISM AND MODEL

In this section, we describe the anomaly detection mechanism used in the proposed methodology and also the principled analytical model proposed.

A. ANOMALY DETECTION MECHANISM

The anomaly detection algorithm based on performance degradation works by continuously measuring the throughput, $\hat{x}$, and maintaining B buckets of depth D, whose state depends on the history of the most recent throughput values, as shown in Figure 3.

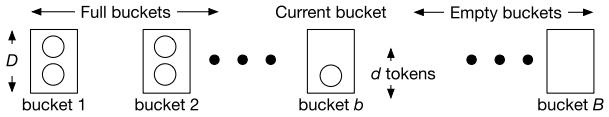


FIGURE 3. Bucket Algorithm dynamics: System of buckets diagram representing the dynamics of the detection algorithm showing *B* buckets of depth *D* each. *d* is the number of tokens in the current bucket, *b*. *D* and *B* must be properly parameterized according to the system under test. Note that a buffer underflow indicates recovery from transient performance degradation, while an overflow of bucket *B* triggers a security alarm.

The scalar value b is a pointer to the current bucket, $b = 1, \ldots, B$, and d is the number of recent throughput samples that deviated from a given target, $d = 0, 1, \ldots, D$. We refer to d as the number of tokens in the current bucket. The total number of tokens in all buckets, in turn, is a proxy for the system degradation level.

Let μ be the baseline average throughput, and σ be the baseline standard deviation. Both μ and σ can be derived from the execution of controlled experiments (**golden runs**). The pointer b to the current bucket is used to determine the current target throughput, which is given by

$$\widetilde{x} = \mu - (b-1)\sigma. \tag{1}$$

After each throughput sample is collected, if its value is smaller than $\widetilde{x}$ the number of tokens in the current bucket is incremented by one.

Note that according to Eq. (1) the target throughput could be negative. To avoid scenarios wherein the target throughput is negative or negligible, i.e., wherein the number of tokens cannot be incremented, we introduce a constant ϵ, that serves as a lower bound on the target throughput value, $\widetilde{x} = \max(\mu - (b-1)\sigma, \epsilon)$. Throughout this paper, in all the scenarios of interest, the right-hand side of (1) is strictly positive and non-negligible. Therefore, we let $\epsilon = 0$.

The rationale behind the target throughput goes as follows. At the initial bucket, the target throughput is μ: any deviation of the throughput to values smaller than μ causes an increase in the number of tokens in the bucket. Then, each additional bucket corresponds to a smaller target throughput. In particular, once the current bucket overflows (or underflows), the target throughput $\widetilde{x}$ is shifted downward (or upward) by one standard deviation.

By decreasing the target throughput as a function of b, as in Eq. (1), the goal is to prevent false alarms. Indeed, adding tokens to buckets becomes more challenging as b grows, i.e., as we move from left to right in Figure 3. Ultimately, when **all** buckets overflow, a performance degradation event is detected, and an alarm is triggered. Alternatively, when **all** buckets underflow, the system has recovered from a transient performance degradation event.

The proposed performance degradation detection algorithm, hereinafter referred to as the Bucket Algorithm (*BA*), is given by Algorithm 1.

The performance degradation detection algorithm can be tuned by varying the bucket depth, D, and the number of buckets, B. The larger the product $D \times B$ the longer it takes for the algorithm to detect the performance degradation. The statistical analysis of the behavior of this family of *BA*s has been described on [13].

Algorithm 1 The Bucket Algorithm

```
b ← 1; d ← 0 { buckets are empty}
for each new throughput sample x̂ do
    if (x̂ < μ − (b − 1)σ) then
        d ← d + 1 { add a token to the current bucket}
    else
        d ← d − 1 { remove a token from current bucket}
    end if
    if (d > D) then
        d ← 0; b ← b + 1 { bucket overflow}
    end if
    if (d < 0 and b > 1) then
        d ← D; b ← b − 1 { bucket underflow}
    end if
    if (d < 0 and b = 1) then
        d ← 0 { recovered from transient degradation }
    end if
    if (b > B) then
        alarm()
    end if
end for
```

B. HYPOTHESIS TESTING

The system administrator continuously considers two alternative hypotheses: (*i*) null hypothesis H_0 corresponding to a situation where there is no attack taking place, and (*ii*) alternative hypothesis H_1 meaning that the system is under attack. Then, the key quantities of interest can be defined as a function of H_0 and H_1. To simplify the presentation, in what follows, time is measured in the number of collected samples.

Definition 1: The mean time until a false alarm under H_0 is denoted by $A_B(D)$.

As discussed in the following section, $A_B(D)$ is given by the mean time to reach the absorbing state of a Markov chain characterizing the bucket algorithm. When $B = 2$, we provide closed-form expressions for $A_B(D)$.

Definition 2: A lower bound on the number of samples until a true-positive under H_1 is denoted by L. Assuming all buckets are initially empty, we let $L = BD$.

Definition 3: The probability of a false alarm under H_0 is the probability that an alarm is triggered outside an attack, $f_B(D) = \mathbb{P}(R < T)$, where R is a random variable with mean $A_B(D)$ characterizing the time until an alarm is triggered, and T is a random variable with mean $1/\alpha$ characterizing the time until an attack occurs.

In this paper, except otherwise noted, we assume that $f_B(D)$ depends on R and T only through their means.

Definition 4: The expected cost of a given system parameterization is a weighted sum of the probability of false-positives, computed under H_0, and a lower bound on the number of samples to detect an attack, computed under H_1,

$$C(\boldsymbol{p}, w, D, B, \alpha) = BD + wf_B(D) \tag{2}$$

TABLE 1. Table of notation.

variable	description
	Basic Terminology (Section IV-A)
$\hat{x}$	current sample
μ	baseline mean of the metric of interest, e.g., baseline average throughput or response time
σ	baseline standard deviation
$\widetilde{x}$	target value of metric of interest
B	number of buckets
b	current bucket, $b = 1, \ldots, B$
D	maximum bucket depth
d	current depth of bucket b, $d = 0, \ldots, D$
	Bucket Algorithm Modeling and Analysis (Sections IV-B to IV-E)
$A_B(D)$	mean time to a false alarm, under the hypothesis of no attack (mean number of collected samples to reach absorbing state)
$f_B(D)$	probability of false alarm
F	target probability of false alarm
α	attack rate
p_i	probability that a sample adds ball to the bucket when $b = i$
	Unified Framework for Sequential Analysis (Section IV-F)
X_n	n-th sample
S_n	current state of sequential analysis
$S^{(l)}$	lower bound on system state (absorbing barrier)
$g(X_n)$	incremental additive contribution of X_n to S_n

Table 1 summarizes the notation introduced in this section. Additional details on how to estimate $A_B(D)$ and $f_B(D)$ are provided in Sections IV-C and IV-D, respectively. The cost function (2) (Definition 4) will be instrumental to parameterize the bucket algorithm in Section IV-E.

C. ANALYTICAL MODEL

Simple algorithms to detect attacks, such as the *BA*, can be tuned using first principles.

The larger the depth of the bucket, the lower the false-positive probability, but the longer it takes for a true-positive to be identified. To simplify the analysis, we **work under the assumption** that anomalies, e.g., caused by attacks, will change the throughput distribution, and will always be detected. However, the number of samples to detect the attack may vary depending on the depth of the bucket. Our **second key simplifying assumption** is that the number of samples to detect an anomaly is much smaller than the number of samples collected before getting a false-positive (which should be always the case in practice: the time until a false-positive should be much longer on average than the time until a true-positive). These are acceptable assumptions as the target of the methodology are issues that affect the system performance and also considering that false-positives will only be caused by unexpected changes in the operation profile, which are not frequent.

We aim at answering the following question: *what is the smallest bucket depth to produce a false-positive probability bounded by a given threshold*?

In the following, we introduce a discrete-time birth-death Markov chain (DTMC) to characterize the behavior of the *BA*. State (b, d) of the Markov chain corresponds to the setup wherein there are d balls in bucket b, and D balls in buckets $b-1, \ldots, 1$.

Each transition of the DTMC corresponds to the collection of a new sample. Such a sample causes the system to transition from state (b, d) to one of its two neighboring states. Let p_i be the probability that the number of balls at bucket i increases after a new sample is collected. Then, $p_i = \mathbb{P}(\hat{x} < \mu - (i-1)\sigma | b = i)$, for $1 \le i \le B$. We denote the vector of such probabilities by $\boldsymbol{p}$, where the i-th element of vector $\boldsymbol{p}$ equals p_i. The entries of the transition probability matrix are readily obtained from Figure 4.

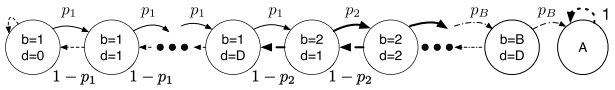


FIGURE 4. Discrete time Markov chain characterizing the behavior of the *BA*. Each transition corresponds to the collection of a new sample.

Once the terminal absorbing state is reached an alarm is triggered. The DTMC is illustrated in Figure 4. The number of samples collected until absorption accounts for a tradeoff between the mean time until (a) a false alarm, in the absence of attacks, and (b) a detection, in the presence of an attack. Larger values of bucket depth D favor the reduction of the former but increase the latter.

Let $\tilde{A}_B(D; \boldsymbol{p})$ be the time until absorption, measured in the number of collected samples, accounting for B buckets of depth D each. We denote its mean by A_B, $\mathbb{E}(\tilde{A}_B) = A_B$. Under the hypothesis of no attack, $\tilde{A}_B$ is the time for a false alarm. We derived a closed-form expression for A_B (Appendix B-A), which is instrumental to handle tradeoffs in the choice of the bucket depth D as illustrated in the upcoming sections. In particular, for $B = 2$, the resulting expression is given by

$$A_2(D; p_1, p_2) = A_2^{(1)}(D; p_1) + A_2^{(2)}(D; p_1, p_2) \tag{3}$$

where $A_2^{(1)}$ and $A_2^{(2)}$ represent the average time required to add balls to the first and second buckets, respectively, before an overflow. Specifically,

- $A_2^{(1)}$ represents the average time taken to add $D+1$ balls to the first bucket, starting from an empty system.
- $A_2^{(2)}$ represents the average time taken to add D balls to the second bucket, starting from a system where the first bucket is full, and the second bucket initially contains a single ball.

Note that once $D + 1$ balls are added to the first bucket, it will overflow, resulting in the second bucket containing a single ball. Furthermore, if additional D balls are added to the second bucket, it will also overflow, triggering an alarm.

Then,

$$A_2^{(1)} = \Delta_1 \left(\delta_1^{(D+1)} - (D+1) \right) \tag{4}$$

$$A_2^{(2)} = \Delta_2 \left(\delta_2^{(D)} - D \right) + \Delta_1 \left(\frac{1 - \rho_1^{D+1}}{\rho_1^{D+1}} \right) \delta_2^{(D)} \tag{5}$$

and

$$\rho_i = \frac{1}{p_i} - 1, \quad \Delta_i = \frac{1+\rho_i}{1-\rho_i} = \frac{1}{2p_i - 1}, \quad \delta_i^{(D)} = \frac{1 - \rho_i^{-D}}{\rho_i - 1}. \tag{6}$$

Note that if $\rho_1 = \rho_2 = \rho$ and $p_1 = p_2 = p$ then the algorithm behavior is equivalent to that with a single bucket, $B = 1$, with

depth $2D$,

$$A_1(2D; p) = A_2(D; p, p) = \Delta_1 \left(\delta_1^{(2D+1)} - (2D+1)\right). \quad (7)$$

Our experimental results indicate that $B = 2$ suffices in the considered scenarios (see Section V). For this reason, in the remainder of this paper, all numerical results derived from the proposed analytical model are reported letting $B = 2$, making use of equations (3)-(6).

D. MODELING THE PROBABILITY OF FALSE ALARMS

We leverage the proposed model to estimate the probability of false alarms. To that aim, we assume that attacks arrive according to a Poisson process with rate α. Recall that $f_B(D)$ denotes the probability of a false alarm (Definition 3). In what follows, we derive expressions for $f_B(D)$ under different assumptions on the distribution of $\tilde{A}_B(D)$.

Assuming that $\tilde{A}_B(D)$ can be roughly approximated by a constant and that the time between attacks is exponentially distributed with mean $1/\alpha$, then

$$f_B(D) = e^{-A_B(D)\alpha}. \quad (8)$$

Alternatively, if we approximate $\tilde{A}_B(D)$ by an exponential distribution,

$$f_B(D) = \frac{1/A_B(D)}{1/A_B(D) + \alpha} = \frac{1}{1 + A_B(D)\alpha}. \quad (9)$$

In the expressions above, we made the dependence of f_B and A_B on the bucket depth D explicit as one of our goals is to study the relationship between D, f_B, and A_B. The closed-form equations (8) and (9) are instrumental to get insights about the interplay between the different model parameters. In particular, as D increases A_B increases and f_B decreases (Definition 1), but the time to detect a real attack increases (Definition 2). As indicated in the sequel, the equations above allow us to find the minimum D such that $f_B(D)$ is below a given threshold. In Section V we experimentally validate that the values of D obtained through the proposed model produce the desired probability of false-positives in realistic settings.

E. PARAMETERIZATION OF THE ANOMALY DETECTION MECHANISM: A MODEL-DRIVEN OPTIMIZATION APPROACH

Next, we show how to use the proposed model and the obtained expressions of the probability of false-positive for the purposes of running statistical hypothesis tests to determine whether there is an ongoing attack in the system.

Given a target false-positive probability, denoted by F, the system administrator's goal is to determine the optimal number of buckets and bucket depth so as to minimize the lower bound on the number of samples to detect and attack, L, while still meeting the target false-positive probability.

Problem with Hard Constraints:

$$\min \quad L = BD \quad (10)$$

$$\text{subject to} \quad f_B(D) \leq F \quad (11)$$

In what follows, we assume that B is fixed and given. Then, as $f_B(D)$ is strictly decreasing with respect to D, the constraint above will be always active and the problem translates into finding the minimum value of D satisfying the constraint. The problem above is similar in spirit to a Neyman-Pearson hypothesis test, for which similar considerations apply, i.e., the optimal parameterization of the test is the one that satisfies a constraint on the false-positive probability.

Alternatively, the problem above can be formulated through the corresponding Lagrangian,

Problem with Soft Constraints:

$$\min \mathcal{L}(D) = BD + w(f_B(D) - F) \quad (12)$$

where w is the Lagrange multiplier. The Lagrangian naturally leads to an alternative formulation of the problem, wherein the hard constraint in (11) is replaced by a soft constraint corresponding to the penalty term $f_B(D) - F$ present in the cost Lagrangian. The Lagrangian is a cost function, motivating Definition 4. Note that as wF is a constant, minimizing (12) is equivalent to minimizing (2).

F. A UNIFIED FRAMEWORK FOR SEQUENTIAL ANALYSIS

Next, we present a general framework for sequential performance analysis. The framework encompasses the proposed *BA* as a special case, allowing the comparison of the considered *BA* against alternative sequential analysis techniques, such as CUSUM, CUSUM-Sign, and SPRT.

In this section, we assume that the collected samples correspond to a metric whose value is such that the lower, the better. Whereas in the remainder of this paper, we consider samples from throughput, in this section we consider, for concreteness and without loss of generality, samples from delay, or inverse throughput. This is in agreement with most of the literature on CUSUM, wherein it is assumed that the metric of interest is such that larger values are worse.

For all the sequential analysis algorithms considered in this paper, we have

$$S_{n+1} = \max(S^{(l)}, S_n + g(X_n)) \quad (13)$$

where S_n is the state of the system after the n-th sample is collected, and X_n is the n-th sample. $S^{(l)}$ is a lower bound on the system state, also known as the process absorbing barrier. Under CUSUM, for instance, $S^{(l)} = 0$. Function $g(\cdot)$ intuitively determines "*how much of an outlier X_n is*." Whenever S_n reaches a target value, an alarm is triggered.

From the above equation, all considered sequential analysis techniques can be considered as random walks, differing on 1) the absorbing barrier $S^{(l)}$, 2) how sample X_n impacts the current state, and 3) the definition of the current state.

Under CUSUM, CUSUM-Sign, and SPRT, the current state is a single scalar value, $S_n \in \mathbb{R}$. Under the bucket algorithm, in contrast, the current state is a discrete vector, characterizing the current bucket and its depth, $S_n \in \mathbb{N} \times \mathbb{N}$. In the case of a single bucket, we have $S_n \in \mathbb{N}$.

With respect to the absorbing barrier, CUSUM and CUSUM-Sign have $S^{(l)} = 0$, whereas SPRT admits a

TABLE 2. Sequential analysis algorithms.

algorithm	barrier, $S^{(l)}$	$g(X_n)$	current state, S_n	comments
CUSUM [51], [52]	0	$X_n - \ell(X_n; H_0)$ given by (14)	$S_n \in \mathbb{R}^+$	cumulative sum, assumes system starts at H_0, and undergoes monitoring of the process until an alarm is raised
SPRT [52], [53]	set as input	log-likelihood ratio given by (15)	$S_n \in \mathbb{R}$	sequential probability ratio test, carrying out a hypothesis test, H_0 versus H_1, over time, possibly raising multiple alarms
CUSUM-Sign [54]	0	$I(X_n - \mu > 0) - \tilde{\kappa}$ given by (16)	$S_n \in \mathbb{R}^+$	CUSUM algorithm adapted to leverage the sign of $X_n - \mu$ so that increment occurs as a function of the event $X_n - \mu > 0$
Bucket [13]	$(0, 0)$	$(f_b(X_n), g_d(X_n))$ given by (21)-(23)	$S_n = (b, d) \in \mathbb{N}^2$	extends CUSUM-Sign to account for 2-dimensional state allowing for time-varying parameter changes

lower absorbing barrier. This implies that for CUSUM and CUSUM-Sign we have $S_n \geq 0$ whereas for SPRT we have $S_n \in (-\infty, +\infty)$.

CUSUM and SPRT may differ on how samples impact the current state, but it is typically assumed that

$$g(X_n) = X_n - \ell(X_n; H_0) \tag{14}$$

and

$$g(X_n) = \log \ell(X_n; H_1) - \log \ell(X_n; H_0) = \log \frac{\ell(X_n; H_1)}{\ell(X_n; H_0)}, \tag{15}$$

for CUSUM and SPRT, respectively. Here, $\ell(X_n; H_0)$ and $\ell(X_n; H_1)$ denote the likelihood of X_n given hypothesis H_0 and H_1, respectively. Intuitively, S_n increases if X_n is more likely under the hypothesis of an anomaly as opposed to the null hypothesis, i.e., if $\ell(X_n; H_1) - \ell(X_n; H_0) \geq 0$.

CUSUM-Sign differs from CUSUM by using an indicator variable $I(X_n - \mu > 0)$ to determine whether the current state should be increased, where $I(c)$ equals 1 if condition c holds, and 0 otherwise. Under CUSUM-Sign we have

$$g(X_n) = I(X_n - \mu > 0) - \tilde{\kappa} \tag{16}$$

where $\tilde{\kappa} = p_0 - \hat{\kappa}$, p_0 is a baseline estimate of the probability that $X_n - \mu > 0$, e.g., due to random noise, and $\hat{\kappa}$ is a constant.

CUSUM-Sign resembles the *BA* as both have a discrete component to determine whether the current state must be incremented or not. However, they differ in a number of aspects, including the fact that CUSUM-Sign parameters are homogeneous over time, whereas the *BA* admits a change in its parameters as a function of the current bucket, providing additional flexibility in the search for anomalies.

Under the *BA*, let $S_n^{(b)}$ and $S_n^{(d)}$ be the bucket index and bucket depth at the n-th iteration of the algorithm. Then, state S_n is given by an ordered pair,

$$S_n = (S_n^{(b)}, S_n^{(d)}). \tag{17}$$

Correspondingly, the dynamics of S_n are governed by two functions, $g_b(X_n)$ and $g_d(X_n)$, which impact the first and second coordinates of the ordered pair. In particular,

$$g_d(X_n) = \text{Sign}(X_n - \mu') + \kappa'(X_n) \tag{18}$$

where

$$\text{Sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

Note that μ' and $\kappa'(X_n)$ play, in the *BA*, the roles of μ and $-\tilde{\kappa}$ in the CUSUM-Sign algorithm, respectively. Indeed, μ' is related to μ as follows,

$$\mu' = \mu + (S_n^{(b)} - 1)\sigma \tag{20}$$

and

$$\kappa'(X_n) = \begin{cases} -(D+1), & \text{if } S_n^{(d)} + \text{Sign}(X_n - \mu') = D + 1 \\ D + 1, & \text{if } S_n^{(d)} + \text{Sign}(X_n - \mu') = -1 \\ 0, & \text{otherwise.} \end{cases} \tag{21}$$

In addition,

$$g_b(X_n) = \begin{cases} +1, & \text{if } S_n^{(d)} + \text{Sign}(X_n - \mu') = D + 1 \\ -1, & \text{if } S_n^{(d)} + \text{Sign}(X_n - \mu') = -1 \\ 0, & \text{otherwise.} \end{cases} \tag{22}$$

The two functions above together comprise $g(X_n)$ for the *BA*,

$$g(X_n) = (g_b(X_n), g_d(X_n)) \tag{23}$$

and

$$S_0 = S^{(l)} = (0, 0). \tag{24}$$

Comparing the CUSUM-Sign dynamics against the *BA*, we note that both rely on the sign of X_n minus a constant. However, as observed in (16), CUSUM-Sign produces a real scalar as its state, whereas the *BA* produces a discrete vector (23) leveraging the sign of $X_n - (\mu + (b-1)\sigma)$ to determine whether depth should be incremented or decremented.

Recall that under the *BA* we refer to $A_B(D; p_1, p_2)$ as the mean time until a false alarm, accounting for B buckets of depth D each. In the CUSUM terminology, A_B is referred to as the average run length (ARL). According to [51], "it captures the average number of articles sampled before action is taken." Under the hypothesis that the system is initially not facing anomalies, the larger the value of ARL, the longer it takes for the system to produce a false alarm.

V. EXPERIMENTAL VALIDATION

To illustrate and validate the methodology described in Sections III and IV, we ran an experimental campaign using the TPC Express Benchmark V [55] (TPCx-V). We emulated the effects of security intrusions that affect performance through a fault injection approach, as described in Section V-A. Then, in Section V-B we report on the exploratory analysis, profiling, and operational phases of the considered framework.

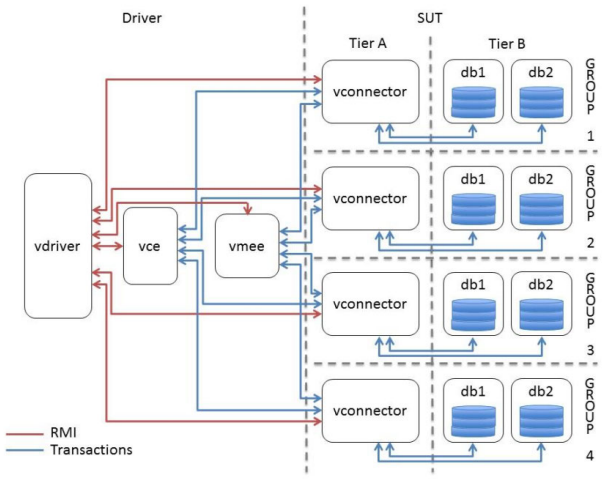


FIGURE 5. TPCx-V components and transactions flow (from [55]). In this work, each group is treated as a distinct subsystem.

Finally, Section V-C assesses the proposed model-based calibration of the anomaly detection system, considering the system under test, and comparing it against CUSUM.

A. SYSTEM UNDER TEST

Next, we introduce the system under test. We start by presenting the TPCx-V workload (Section V-A1). Then, we describe our experimental setup (Section V-A2) leveraging a fault model (Section V-A3) on top of TPCx-V.

1) TPCx-V WORKLOAD

TPCx-V is a publicly available, end-to-end benchmark for data-centric workloads on virtual servers. The benchmark kit provides the specification, implementation, and tools to audit and run the benchmark. Details can be found in [55]. It models many features commonly present in cloud computing environments, such as multiple VMs running at different load demand levels and significant fluctuations in their load level [14].

We use the workload and software provided by the TPCx-V to emulate the context of a real-world scenario of brokerage firms that must manage customer accounts, execute customer trade orders, and be responsible for the interactions of customers with financial markets [56].

Figure 5 shows the transaction flow. The virtual client emulator (vce), interacts with the different brokerage firms (distinct groups), which in turn communicates with the virtual market emulator (vme). TPCx-V uses virtualization technology to co-locate database tiers and application-management tiers on logically distinct VMs within a single computer system.

The goal of TPCx-V is to measure how a virtualized server runs database workloads, using them to measure the performance of virtualized platforms. The minimal deployment of the TPCx-V has four groups, each with three VMs, representing different subsystems. A typical run has *10 distinct* load phases of *12 minutes* each. The different components of TPCx-V are architecturally distributed as depicted in Figure 5.

The TPCx-V workload is made up of *12 types of transactions* with different characteristics used to simulate the stock trade process. They are submitted for processing at multiple databases (market, customer, and broker) following a specified mix of transactions for different phases. The main performance metric for the benchmark is the business throughput (`tpsV`). It represents the number of completed *Trade-Result* per second.

The TPCx-V workload provides an adequate testbed environment for our anomaly detection approach, since it captures the scalable nature of complex virtualized environments by providing different groups of virtual machines with different sizes and configurations while serving an elastic workload in different phases of the execution.

2) EXPERIMENTAL SETUP

Our setup is a deployment of the TPCx-V over two physical servers. The first server is a Dell PowerEdge R710 with 24 Cores, 96GB RAM, and 12TB disk, and is managed by a Xen hypervisor (4.4.1). It has a privileged domain (dom0), and 17 virtual machines with different configurations, a set dedicated to the TPCx-V, and another set that will represent our compromised tenants. The second server is configured with 2 Cores, 8GB RAM, and 1 TB disk, with the same Xen hypervisor (4.4.1). It runs the VM driver component into a different host as prescribed on the TPCx-V specification. The details of each VM are described in Table 3 together with the resource specifications for each group. *gn* refers to the VM for group *n*. Each group was defined according to the benchmark recommendations [55]. As a group is a set of three VMs, our malicious user will have access to the same amount of VMs. Note that we are overcommitting the number of vCPUs (see Table 3) issuing 45 vCPU, which is greater than the physically available number of cores. This is a common strategy in cloud computing to optimize resource usage [8] since not all vCPU is fully used at the same time.

We also developed a management tool responsible for triggering all tests while monitoring the physical environment. This management tool captures events, reports any problem during the test, and handles all interactions between the environment, benchmark, and tests. Each **single experiment lasts roughly 4 hours** that correspond to the 2h demanded as a minimum by the benchmark specification, and another 2h to restore the full environment to its initial state. Initial state restoration is achieved by rebooting the servers and recovering the system and databases (restoring all virtual disks).

The vanilla configuration of the TPCx-V is designed to stress the system and evaluate the maximum load the virtualized can handle. However, this workload is not representative of common operational services. To handle this limitation, we changed the default configuration so that carried load is just a fraction of the system capacity. Otherwise, any other activity on the system could jeopardize the validity of our

TABLE 3. VMs name, memory, and the number of virtual CPUs. The tpc-driver is supported on a different physical host.

vm	GB	vCPU	vm	GB	vCPU	vm	GB	vCPU
g1a	1	1	g1b2	4	4	g1b1	8	2
g2a	1	1	g2b1	12	2	g2b2	6	6
g3a	1	1	g3b1	16	2	g3b2	8	8
g4a	1	2	g4b1	20	2	g4b2	10	8
Ten. A	1.5	2	Ten. B	1.5	2	Ten. C	1.5	2
Dom0	1.9	4	Driver	1.8	2			

experiments since we are not accounting for transient spikes on the carried load.

3) FAULT MODEL

The proposed fault model abstracts from a common cloud computing attack pattern: the resource exhaustion pattern [29], where a virtual guest can obtain more resources than allowed. This pattern can be categorized as follows [57]: i) *excessive use*, where there is no abnormal use, but the consumption of resources is significantly higher for one tenant, and ii) *malicious use*, where the malicious excessive use of resources can cause a failure.

Note that the resource exhaustion pattern is one of many possible workloads that can be assessed using our technique. In particular, any attack that affects the overall system performance could be evaluated using our approach. Still, we adopted the resource exhaustion pattern since we could fine-tune its intensity and frequency to enrich the evaluation, which may not be easily possible with other workloads. Nevertheless, our approach does not cover attacks with no performance impact, having little or no effect on detecting them.

TPCx-V is a database-centric benchmark, and thus an attack that explores database resources can impact performance. However, we are not aware of a documented exploit focused explicitly on the hypervisor that attempts to exhaust the resources used by database services. This gap motivated us to use Stress-NG [58] to simulate resource exhaustion behavior. Stress-NG exercises computer subsystems and operating system kernel interfaces. Hackers produce malware [59] using the same kernel interfaces as Stress-NG.

We have defined three configurations to emulate the resource exhaustion attack:

(H): A *High*-Intensity workload: starts eight processes to exercise the system IO and runs for 300 seconds;

(L): A *Low*-Intensity workload: perform ten intervals of 15secs of IO-exercise and 15s with no workload. The workload uses two IO stressors processes and runs for 300s;

(Ls): Shorter Low-Intensity workload: the same as the (L) configuration but with only three intervals.

We defined the configuration attack length based on the proportional time of the TPCx-V run, about 4% and 1%. The configuration length is also less than the benchmark phase (*12 minutes*, as explained in Section V-A).

Since the TPCx-V has different phases with diverse load demands (see Section V-B3), we focused the attack on two distinct phases, on the *4th*, which is when **the group with more physical resources has a more significant contribution to the overall load** and the *6th*, when **the reference metric achieves the highest rate**.

Combining those two definitions, we have a total of `6` fault models, which we will refer to using the phase plus the configuration reference: `4H`, `4L`, `4Ls`, `6H`, `6L`, and `6Ls`.

B. EXPLORATORY, PROFILING AND OPERATIONAL PHASES

In the following three sections, we revisit the exploratory, profiling, and operational phases. Those three phases were previously introduced in Sections III-A, III-B and III-C, respectively, and are now analyzed in light of the system under study.

1) EXPLORATORY PHASE

In this step, we performed a transaction characterization using exploratory runs, leveraging *BA* and TPCx-V run data. The analysis revealed that not all of the TPCx-V transactions are impacted by the system load, i.e., the throughput of some transactions does not vary as the system degrades. To conclude the exploratory phase, we defined our monitoring surface as the throughput information of *9* (of the *12*) transactions from TPCx-V. That data was evaluated distinctly for every subsystem (the *4* TPCx-V groups), resulting in 36 (9×4) *BA* instances running in parallel. For each of the *BA* instances, we associate *10* pairs of throughput mean and standard deviation, *1* for each distinct operation profile (*10* TPCx-V phases). Each pair of parameters corresponds to 12 minutes of continuous operation of the benchmark (see Section V-A).

2) PROFILING PHASE

To profile the system, we executed golden runs to generate data for the characterization of the baseline behavior of the system (37 golden runs, comprising the profiling set, or *P set*) and for validation (22 golden runs, comprising the validation set, or *V set*). For every transaction from the monitoring surface, we computed and stored the average throughput for each subsystem in every operational profile. These values are the baseline metrics. To calibrate our performance model (Section V-C), we need to account for the following metrics:

1) The probability of a false-positive alarm as a function of bucket depth;
2) The probability for each transition in the DTMC (Figure 4);
3) The mean time to first alarm during an attack.

To compute those probabilities, we applied the *BA* with different configurations over the validation runs. These results are presented in Section VI.

During the validation process, we implemented the *BA* for each golden run, taking into account the number of

TABLE 4. False-positive alerts: total counts over validation runs.

B	D	FP alerts	B	D	FP alerts	B	D	FP alerts
1	30	27,373	2	12	4	2	21	0
2	6	185	2	15	0	3	10	0
2	9	12	2	18	0			

TABLE 5. False-positive alerts segmented by TPCx-V's transactions over validation runs.

Bucket depth (D) \ Transaction ID	0	1	2	4	5	6	7	8	9
6	23	43	28	34	12	23	14	2	6
9	0	1	4	6	1	0	0	0	0
12	0	0	2	2	0	0	0	0	0

false-positive alerts. The main objective of this step was to assess whether utilizing the model-recommended parameters would lead to an acceptable level of false-positive alerts.

Initially, we conducted the assessment using the same runs employed to generate the baseline metrics (referred to as the *B set*). This phase did not involve validation; rather, it aimed to generate values for later comparison with the validation results. Additionally, this step allowed us to ensure consistency and identify potential computational errors by analyzing any unusual patterns in the number of alerts.

Subsequently, we performed the same process using the validation runs (referred to as the *V set*). Upon comparing the results from both sets, we found no significant differences.[1] Consequently, in Table 4 we solely present the values obtained from the validation runs.

The results described in Table 4 show that using two buckets is effective, as it produces a fair number of false-positive alerts. Employing a single bucket leads to excessive alerts, while utilizing three buckets is unnecessary, as employing two buckets already results in a sufficiently low number of alerts.

Table 5 shows that different transaction types exhibit varying levels of sensitivity to the same *BA* parameterization. These distinct sensitivities indicate that D maybe need to be adjusted as a function of the transaction type being considered.

In Section V-C we report results on the calibration of D and B using the proposed Markov model as well as the collected measurements described above. Based on the calibrated performance model, it is recommended to configure the bucket depth D within the range of (12, 15] as the number of false-positive alerts within this range remains at an acceptable level, in agreement with the above findings.

3) OPERATIONAL PHASE

As prescribed in Section III-C, we have to define our alert reporting criteria. We adopted distinct approaches to detect

[1] In addition, when running the *BA* in both golden runs sets, we also obtain similar results. This similarity indicates that the profile derived from the baseline metrics can be generalized for different runs.

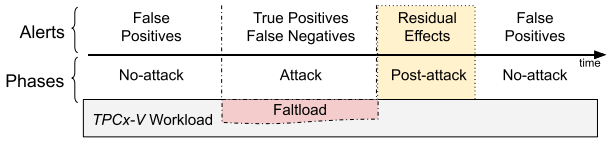


FIGURE 6. Distinct phases and alerts issued during a test run.

TABLE 6. Runs in experimental campaign.

Test	4H	4L	4Ls	6H	6L	6Ls	Golden
Number of runs	21	21	21	21	21	21	59

true-positives (TPs) and *false-positives* (FPs). A TP occurs when we detect **at least one alert on the attack phase**. We only need one alert, in any group, for any transaction type. Complementary, a *false-negative* (FN) occurs when **no alert is raised** on the considered attack phase. We consider an FP **every bucket overflow that occurs in the no-attack phase**. In this scenario, if two buckets overflow, whether in separate transactions or within the same transaction but in different groups, they will be treated as two distinct false positives. This approach is justified as alarm systems aim to minimize the occurrence of false positives.[2]

During the testing campaign process, each of our test runs is composed of three phases, as depicted in Figure 6. In those defined periods, we count the number of FP, TP, and FN as described above. Each alert on the post-attack phase will count as a *residual effect*. When executing the TPCx-V workload, we will run just one attack, as defined in Section V-A3. As shown in Table 6, we performed 21 runs for each fault injection type, while applying the *BA* throughout the monitoring surface.

Following the execution of the tests, we utilized the collected data to assess the performance of *BA* with different parameterizations and examine its effectiveness. Table 7 shows the fraction of true positives over the total number of alerts, which align with the findings presented in Table 4, suggesting that the optimal number of buckets for the given scenario is two ($B = 2$). Section VI delves into a comprehensive discussion of the *BA* results, highlighting its effectiveness across all considered configurations.

C. MODEL ASSISTED CALIBRATION OF ANOMALY DETECTION

In this section we illustrate how to calibrate the bucket algorithm in the realm of the system under test (Section V-C1), and then compare it against CUSUM (Section V-C2).

1) BUCKET ALGORITHM CALIBRATION

This section provides insights into experimental results using the proposed model parametrized with the system's data.

First, we parameterize the proposed model from the experimental data. To exemplify the general process, we focus on the TRADE_LOOKUP transaction. Recall that p_1 =

[2] An alert on the post-attack phase can be an FP or a residual effect caused by the faulty load. This question will be discussed in Section VI-A.

TABLE 7. Fraction of alerts occurring during active attacks (TP), over all alerts, varying *B* and *D*. Note that while Table 4 was generated using validation runs, here we use test runs. In addition, in Table 4 we count residual effects as FP, while here we filter residual effects using the δ-threshold strategy described in Section VI-A.

B	D	TP %	B	D	TP %	B	D	TP %	B	D	TP %
1	30	7.71%	2	9	97.40%	2	15	98.01%	2	21	97.62%
2	6	92.22%	2	12	97.74%	2	18	98.01%	3	10	0.00%

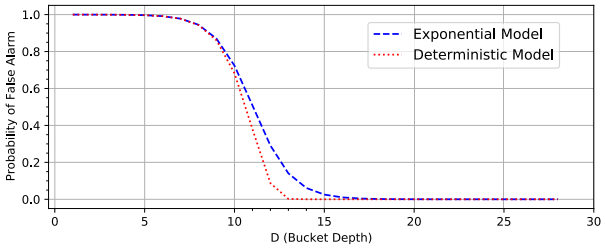


FIGURE 7. Probability of false alarm from model tuned based on experiments.

$\mathbb{P}(\hat{x} < \mu | b = 1)$ and $p_2 = \mathbb{P}(\hat{x} < \mu - \sigma | b = 2)$. Then, we identify that for TRADE_LOOKUP we have p_1 and p_2 equal 0.466 and 0.714, respectively. Interestingly, $p_2 > p_1$, i.e., conditional on the fact that the second bucket has been reached, the probability that the sampled throughput is smaller than $\mu - \sigma$ is larger than the probability that the sampled throughput is smaller than μ while in the first bucket. Indeed, once the second bucket is reached the rate at which tokens are added to the bucket increases, suggesting the need for mechanisms to avoid false alarms. Such observation further motivates a decrease in the target throughput value as a function of b, as discussed in Section IV-A.

We assess the expected number of samples until a false alarm, obtained from (3), with $B = 2$, $p_1 = 0.466$, $p_2 = 0.714$, and letting D vary between 1 and 30. For $D = 15$, we observed that the number of samples until a false alarm surpasses 10^7.

Figure 7 accounts for an attack model, wherein the mean time between attacks is $1/\alpha = 5 \times 10^5$ samples, i.e., the attack rate is $\alpha = 2 \times 10^{-6}$ attacks per sample. As the bucket depth increases, the probability of false alarms decreases. For $D \geq 12$, the probability of a false alarm is close to 0.

As discussed above, there is a tradeoff between the probability of false alarms and the time to detect attacks once they occur. To cope with such a tradeoff, we consider both approaches introduced in Section IV-E, namely the hard and soft constraint problems. Under the hard constraint problem, a target probability of a false alarm is determined, and the minimum value of D that satisfies such a target is sought. For instance, if we set $F = 0.03$ in (11) then the minimum value of D satisfying the constraint is $D = 13$ and $D = 15$ under the deterministic and exponential attack models, respectively.

We also assess how the cost $C(\boldsymbol{p}, w, D, B, \alpha)$ introduced in Definition 4 varies as a function of D, letting $B = 2$, $p_1 = 0.466$, $p_2 = 0.714$ and $\alpha = 2 \times 10^{-6}$. Letting $w = 20.646$ which corresponds to the Lagrange multiplier of the constrained problem under the deterministic model (see also (12)), the optimal bucket depth equals $D = 13$, which is in agreement with the result presented in the previous paragraph.[3] Alternatively, under the exponential model we let $w = 75.239$ to obtain an optimal bucket depth of $D = 15$, again in agreement with the previous paragraph.

Take away message and engineering implications: the analysis presented in this section is instrumental in performing what-if counterfactual analysis and executing utility-driven model parameterization. If the system administrator implements global countermeasures against attacks, for instance, it is expected that the rate of attacks will decrease. In that case, the bucket depth can be adjusted accordingly, e.g., using the proposed utility-driven approach presented in this section.

2) CUSUM COMPARISON

We opted to evaluate the CUSUM [51], [52] method to contrast our approach with traditional sequential analysis algorithms. To that aim, the first step consists in transforming our throughput measurements into a metric for which large deviations above the mean correspond to anomalies (recall from Section IV-F that CUSUM detects large deviations above the mean). Given a throughput x, we experimented with different transformations to produce our target metric x', including e^{-x}, $1/x$, $1/\log(x+1)$ and $1/\sqrt{x}$. All transformations produced similar results. In what follows, we report results for $x' = e^{-x}$.

We consider a vanilla parameterization of the CUSUM method, to allow for a fair comparison against *BA*. In particular, we adapted the '*detecta*' [60] Python package to evaluate the TPCx-V architecture using 36 sequential tests (4 groups of 9 transactions) and its baseline metrics (see Section III-B). We let $S^{(l)} = 0$, and allow '*detecta*' to set the additional parameters.

We observed that CUSUM raised a large number of false positives, even for the golden runs (not shown in the paper). This limited the ability to compare CUSUM against *BA*. Figure 8 shows an example of CUSUM evaluated over the TPCx-V data. The top chart shows the transformed throughput samples as well as the triggered alerts in red (a sequence of red dots corresponds to a contiguous interval wherein alerts were raised). The bottom chart shows the time series of the cumulative sum of changes (both positive and negative). Each TPCx-V phase is also represented with its respective threshold, proportional to its baseline profile. Specifically, the horizontal lines correspond to $T'\sigma$, where the T' is a threshold factor (an input parameter, set at its default value), and σ (which is computed from the baseline profile).

In Figure 8 we observe that the attack did not trigger an alert (similar behavior was observed across our dataset). Then, the miss-detection is followed by a large number of

[3]Note that 1) the Lagrangian is minimized at $D = 12.39$ and we take its ceil as the optimal bucket depth and 2) the Lagrangian also admits other local minima. If we let $w = 909$, in contrast, the optimization problem (12) admits a unique solution, at $D \approx 13.3$, and in this case, we need to take its floor to satisfy $f_B(D) \leq F$.

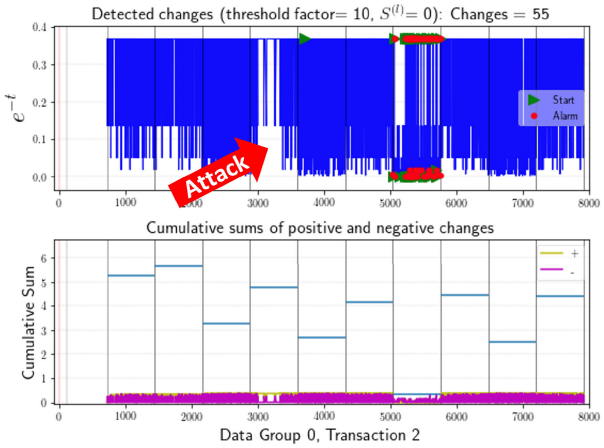


FIGURE 8. An instance of the CUSUM evaluation for the transactions (TRADE_LOOKUP) of the Group 0 of the TPCx-V in a run with an attack in the fourth phase.

false alarms, leading us to conclude that the **direct application** of the CUSUM algorithm to assess anomalies in a complex environment such as TPCx-V is not adequate. Next, we further detail some of the reasons for the low accuracy.

1) input transformation: we transformed throughput data into a target metric whose large deviations above the mean should be avoided. Although we tried four transformations leading to similar conclusions, additional experiments are necessary to determine if there are alternative transformations more suitable to our needs;
2) threshold and absorbing barrier: the parameterization of CUSUM threshold and absorbing barrier are also subject to transformations and may require additional refinements;
3) abrupt changes in input: the frequent and abrupt changes of throughput under TPCx-V negatively impact anomaly detection. In particular, note that the plot in Figure 8(a) corresponds to a line, and we see a whole area filled in blue due to the erratic behavior of the throughput under short time scales. We envision that such abrupt changes can be attenuated by an additional mechanism, such as a moving average.

For the above reasons, the application of the CUSUM methodology in the considered systems requires further research, which we leave as a subject for future work. In particular, we were unable to find a unified parameterization for the CUSUM algorithm that works under all considered workloads and phases, whereas for the *BA* we were able to find a combination of bucket width and depth that reached our goals, as further detailed next.

VI. RESULTS

In this section, we report our experimental results. Our goals are to 1) investigate the role of residual effects after attacks (Section VI-A) and 2) understand the impact of parameters on false positive rate (Sections VI-B to VI-D).

We begin by presenting the residual effects observed after the attack phase. Understanding those residuals allowed us to establish a threshold to avoid false positives, trading between those false positives and the mean time to the first alarm. We then performed three case studies to evaluate how different parameters, such as bucket depth, affect the method's performance. We observed, for instance, that the bucket depth has a significant impact on the false positives, with better results obtained for $D = 12$. Finally, we assessed the robustness of our results through variability tests, which indicated a low variability in performance with a margin of error below 1%.

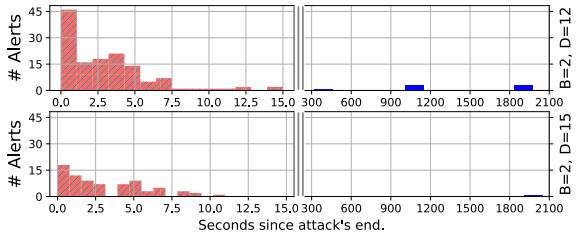


FIGURE 9. Post-attack alerts distribution for bucket configuration with $B = 2$ and $D = [12, 15]$. For readability, the x scale is cropped between 15 and 300.

Overall, our experiments demonstrate the effectiveness and potential of our proposed method in detecting anomalies in cloud transactions, highlighting the importance of appropriate parameter selection and transaction-specific configurations for achieving optimal performance.

Our results are discussed using three metrics widely adopted in classification assessment [61], namely, precision, recall, and F-measure, which are defined as a function of true-positives (TP), false-positives (FP) and false-negatives (FN) as follows,

$$\text{Pr} = \frac{TP}{TP + FP} \quad \text{Re} = \frac{TP}{TP + FN} \quad \text{F1} = \frac{2 \times \text{Pr} \times \text{Re}}{\text{Pr} + \text{Re}}. \tag{25}$$

Precision (Pr) measures the impact of FP on the method's positive prediction. Recall (Re) reflects the sensitiveness of the algorithm, capturing the fraction of corrected predictions. F-measure (F1) is the harmonic mean of precision and recall, balancing them in a single metric.

A. RESIDUAL EFFECTS

Throughout our experimentation, we noted a substantial increase in the number of alerts immediately following the attack phase compared to non-attack periods. To investigate this phenomenon, we conducted an analysis considering the count of alerts and the time interval, measured in seconds, from the conclusion of the attack injection phase to each individual alert. The objective was to comprehend the underlying reasons and mechanisms behind the occurrence of alerts during the post-attack phase.

Figure 9 shows that most of the bucket overflows occur a few seconds after the attack, which suggests that those alerts can be residual effects that should be distinguished from false-positives.

TABLE 8. Mean time to first alarm during the attack injection (in seconds).

Transaction	$D = 6$	$D = 9$	$D = 12$	$D = 15$
TRADE_LOOKUP	31.03	50.06	69.18	61.63
MARKET_WATCH	36.08	54.11	60.38	59.51

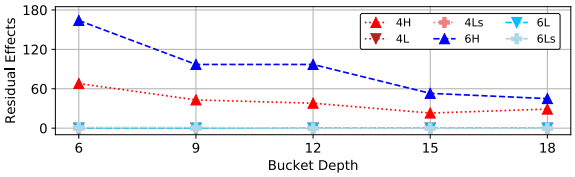


FIGURE 10. Distribution of the residual effects by failure mode and bucket depth.

What is a reasonable threshold that would discriminate between false-positives and residual effects of the attack? Regression techniques, and outliers detection techniques, among others, can be used to estimate that threshold. In this work, the meantime to the occurrence of the first alarm during the attack phase is used as the discrimination threshold.

Let δ be the mean time for *BA* to trigger an alarm following the initiation of an attack. Note that δ is a function of multiple parameters, including B and D. After an attack ends, we assume that any alert occurring within a time frame $t < \delta$ corresponds to residual effects from the preceding attack (such as emptying queues and recovering from error states).

Table 8 shows the mean time to raise an alarm, i.e., to detect an attack, after the attack is issued, for two transactions, TRADE_LOOKUP and MARKET_WATCH, and $D \in \{6, 9, 12, 15\}$. It indicates that δ grows as a function of D. Such growth in δ, in turn, is correlated with a decrease in the number of residual effects, as shown in Figure 10. Figure 10 shows the number of residual effects as a function of D. It indicates that the number of residual effects decreases as a function of D. Indeed, the larger the value of D, the higher the tolerance for transient faults, and the lower is the number of residual effects. Together, Table 8 and Figure 10 provide evidence that δ can be used as a proxy for the time during which alerts after an attack are residual effects from the preceding attack.

Figure 10 also shows that the failure mode with higher intensity (H) triggers the larger number of residual effect alerts. In addition, the number of alerts triggered in the sixth phase (blue dashed line) is greater than the number of alerts triggered in the fourth phase (red dotted line).

What is the reaction time of the proposed approach? We evaluate how fast the proposed approach responds to attacks. Figure 11 shows the frequency of alerts and the cumulative distribution function (CDF) of reaction time. In the horizontal axis, we have the time in seconds since the beginning of the attack. While we previously introduced δ as the average reaction time in seconds, we now present the complete CDF of reaction times, indicating the duration from the beginning of an attack to the triggering of the initial alert by the *BA*.

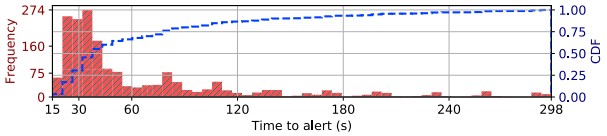


FIGURE 11. The overall distribution of the time to first alert in the presence of an attack. The figure accounts for samples from all fault models and configurations.

Recall that in our experiments samples are collected at a rate of one sample per second. Therefore, assuming that all buckets are empty in the beginning of the attack, the reaction time is lower bounded by *BD* seconds, as it takes at least *BD* samples to overflow all buckets (see also Definition 2 in Section IV-B). In our experiments, more than 50% of the anomalies were detected in less than half a minute, and more than 75% were detected in less than a minute. These detection times are quite promising when considering the following factors: 1) our sampling resolution is in seconds; 2) we are evaluating a complex system; and 3) the algorithm is designed to handle temporary faults.

B. CASE STUDY 1

In our initial case study, we examine how the *BA* performs in terms of detection effectiveness across various fault models and analyze the impact of parameterization on its performance. For this particular case study, we employ a consistent set of parameters for all operations. Through calibration of our model, we determine that the optimal value for D falls within the range of 12 to 15. Table 9 reports the metrics obtained from our tests, segmented by the six different failure modes, as well as a combined analysis of all failure modes (line denoted as *All*).

TABLE 9. Result of Case Study 1 showing the Residual effects counts (RE), Precision, Recall and F-measure (F1) metrics. (Maximal value for TP in ALL is 126, others classes is 21).

		TP	*FN*	*FP*	*RE*	*Pr*	*Re*	*F1*
B = 2 & D = 12	All	108	18	12	135	0.90	0.86	0.88
	4H	20	1	0	38	1.00	0.95	0.98
	4L	21	0	3	0	0.88	1.00	0.93
	4Ls	9	12	1	0	0.90	0.43	0.58
	6H	21	0	2	97	0.91	1.00	0.95
	6L	21	0	4	0	0.84	1.00	0.91
	6Ls	16	5	2	0	0.89	0.76	0.82
B = 2 & D = 15	All	82	44	2	76	0.98	0.65	0.78
	4H	20	1	0	23	1.00	0.95	0.98
	4L	16	5	0	0	1.00	0.76	0.86
	4Ls	1	20	1	0	0.50	0.05	0.09
	6H	21	0	0	53	1.00	1.00	1.00
	6L	17	4	1	0	0.94	0.81	0.87
	6Ls	7	14	0	0	1.00	0.33	0.50

The first noteworthy observation pertains to the relatively low number of alerts generated by the method outside an attack. Considering that we account for 4×9 instances of the *BA* running simultaneously (as explained in Section V-B1), the potential count for this category could be significantly higher. Table 9 shows that the usefulness of the *BA* varies depending on the intensity of the attack and the configuration of the algorithm. In most cases and configurations assessed, the F-measure exceeds 0.78. Notably, for the 6H fault model

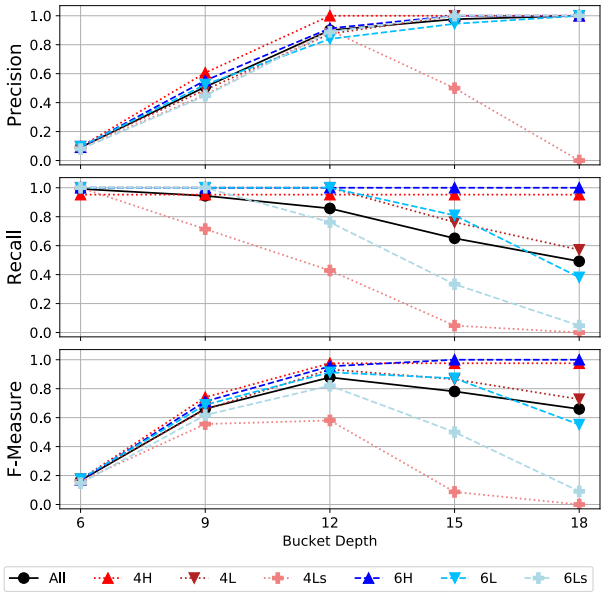


FIGURE 12. Campaign results for all fault models using two buckets. The data are shown with the pre and pos phases split into two sets.

with $D = 15$, the F-measure reached 1. However, we discovered that for short attacks with low intensity, static values of D proved to be less effective. A prime example is the 4Ls fault model with $D = 15$, where we observed an F-measure of 0.09. This can be attributed to the fact that larger values of D are unable to effectively detect shorter bursts.

The impact of the bucket depth D on the target metrics (Equation (25)) is further illustrated in Figure 12. A notable observation derived from the Precision and Recall curves is that these metrics exhibit contrasting trends with respect to the bucket depth. Precision increases as D grows, due to a decrease in false-positives (FPs) as D increases. This can be attributed to the higher tolerance for performance variability under normal conditions that comes with a larger D. Conversely, Recall shows a decrease with increasing D due to a rise in false-negatives (FNs). As D becomes larger, it takes longer to detect attacks, leading to an increase in FNs. Thus, the F-measure, that combines Precision and Recall through their weighted harmonic mean, effectively captures some of the inherent tradeoffs involved in the efficiency of the *BA*.

For the scenario encompassing all failure modes (curve labeled as 'All'), $D = 12$ yields the highest F-measure, in close agreement to $D = 13$ found in Section V-C1. However, the optimal value of D is sensitive to the failure mode and transaction type, and conditional parametrization may yield improvements, as further discussed next.

C. CASE STUDY 2

Next, we aim to further investigate how different transaction types impact optimal parametrization. Recall from Table 5 that the optimal bucket depth varies as a function of the considered transaction. As a result, we anticipate that employing different parameterizations for different transactions could yield more favorable performance outcomes. To verify this hypothesis, we have devised a parameterization based on a mixture (referred to as 'Mix 6/9/12/15'), wherein the bucket depth for each transaction was set as the minimum D between 6, 9, 12 and 15, for which the number of false positives equals zero during the validation runs (see Table 5 for the results obtained during the validation runs). The results depicted in Table 10 indicate that using such a strategy we obtain a higher number of false positives (FP) compared to Table 9, causing a detrimental impact on the overall performance. This is due to the fact that transactions tracked with a lower bucket depth result in a greater number of false positives.

TABLE 10. Result of Case Study 2, showing the residual effects counts (RE), Precision, Recall, and F-measure (F1) metrics (Maximal value for TP in ALL is 126, others classes are 21).

		TP	FN	FP	RE	Pr	Re	F1
Mix 6/9/12/15	All	111	15	33	115	0.77	0.88	0.82
	4H	20	1	3	34	0.87	0.95	0.91
	4L	21	0	8	0	0.72	1.00	0.84
	4Ls	9	12	6	0	0.60	0.43	0.50
	6H	21	0	3	81	0.88	1.00	0.93
	6L	21	0	5	0	0.81	1.00	0.89
	6Ls	19	2	8	0	0.70	0.90	0.79
Mix 12/15	All	106	20	9	114	0.92	0.84	0.88
	4H	20	1	0	34	1.00	0.95	0.98
	4L	21	0	2	0	0.91	1.00	0.95
	4Ls	8	13	1	0	0.89	0.38	0.53
	6H	21	0	1	80	0.95	1.00	0.98
	6L	21	0	3	0	0.88	1.00	0.93
	6Ls	15	6	2	0	0.88	0.71	0.79

Inspired by the analytical model, the bottom half of Table 10 shows the results obtained using an alternative mixture of bucket depths. In this new setup, depths 6 and 9, initially included in the mixture discussed in the preceding paragraph, were removed, resulting in a mixture with depths 12 and 15. Table 10 indicates that after making this adjustment, the count of false positives notably decreases, particularly for the L scenarios. In essence, this findings highlight the potential use of dynamic tuning methods [13] to achieve a balance in algorithm parameters, leading to performance enhancements, and further strengthening the rationale behind the proposed methodology.

D. CASE STUDY 3

For case study 3, we rely on a different type of workload in comparison to case studies 1 and 2. Specifically, we focus on Media Microservices, that involves the deployment of numerous virtual machines within a cloud environment. To assess the performance, we leverage the Media Microservices benchmark provided by the DeathStarBench suite [62].[4]

In the previous case studies, we reported on anomaly detection triggered by security attacks, leveraging the throughput as the metric of interest, while in this case study we report on anomaly detection triggered by a memory leak, leveraging average response time as the metric of interest. Specifically, we evaluate the response time by quantifying its deviation

[4]https://github.com/delimitrou/DeathStarBench/tree/master/mediaMicroservices

with respect to the predefined response time requirement. The considered measure is referred to as the normalized distance and is defined in [63] as follows:

$$\mathscr{D}(m; r) = 2\frac{m}{m + r} \tag{26}$$

where $\mathscr{D}(m; r)$ is the normalized distance, m is the measured response time, and, r is the baseline time requirement. Based on the given definition, when the measured response time matches the requirement, the normalized distance is equal to 1. In systems that meet the performance requirement, the normalized distance will be below 1, and the normalized distance serves as a metric for assessing the average performance evolution. However, in order to account for transient performance changes, we introduce a new metric denoted as the σ-normalized distance that generalizes the normalized distance. This metric incorporates the standard deviation of the measurement and is defined as:

$$\mathscr{D}(m, \sigma; r) = 2\frac{m + \sigma}{m + \sigma + r}. \tag{27}$$

The σ-normalized distance is the normalized distance accounting for a standard deviation: m is the measured response time, σ is the measured standard deviation, and r is the baseline response time requirement.

Inspired by the methodology introduced in this paper for anomaly detection based on deviations of throughput or expected response time, we consider the problem of anomaly detection in the current case study. Here, anomalies are defined as any deviations that exceed 3 standard deviations in relation to the expected response time, corresponding to 3 buckets. Consequently, in the subsequent analysis, the baseline response time requirement r is established as

$$r = \mu_\ell + 3 \cdot \sigma_\ell \tag{28}$$

where μ_ℓ is the average response time under low load, and σ_ℓ is the standard deviation of the response time under low load.

The Media Microservices benchmark supports a load balancer and several database instances to store movies and their reviews. In particular, it characterizes client requests that use Ngnix for load balancing. In this work, we focus on the part of the workload related to the movie review functionality.

1) MEMORY LEAK AND SYSTEM SETUP

This case study involves a memory leak that is an inherent component of the Media Microservices benchmark. As mentioned above, the objective of this case study is to demonstrate the effectiveness of a response time tracking methodology in facilitating anomaly detection. To this aim, we ran the benchmark on a heterogeneous network of virtual machines, with 4Gbytes (large) and 8Gbytes (extra-large) of main memory. The network was composed of a total of 30 virtual machines, with 15 large and 15 extra-large machines.

2) RESULTS

Table 11(a) shows the response time, normalized distance, and σ-normalized distance, for large machines, while Table 11(b) shows the same information for the extra-large machines. Large machines were observed during one week, and extra large machines were observed after several weeks of aging. Anomalies are identified when normalized distance values exceed 1.0. In the case of σ-normalized distance, anomalies are characterized by values surpassing 1.2. For large machines, σ-normalized distance is able to capture some anomalies that were not captured through normalized distance. For extra-large machines, both metrics work equivalently under the considered thresholds.

The presented data showcases the ability of a methodology based on response time tracking to support anomaly detection. In particular, it indicates that we are able to detect anomalies from multiple vantage points relying on the deviation of response time with respect to its reference value. We leave a detailed analysis of the bucket algorithm in additional scenarios such as the one considered in this section as a subject for future work.

VII. DISCUSSION

Next, we discuss some of the assumptions considered in this paper and their implications.

Applicability domain: Our technique mainly applies to anomalies induced by attacks that cause performance deviation. The approach can generalize to any resource degradation that impacts the mean performance, not necessarily implying exhaustion. The rationale behind this is that our approach evaluates the behavior of a system when subjected to anomalous load compared against a baseline. The considered anomalies can be caused by different types of faults, including attacks that exploit vulnerabilities (which may be unknown) affecting performance. Moreover, our approach specifically aims to assess complex systems with long-lived workloads. For isolated, short-lived tasks, one may need to resort to more computationally intensive anomaly detection mechanisms.

Practical significance: One could argue that our approach requires a consistently stable operational load, which can be challenging to find in a real-world system over extended periods. However, our observations indicate that despite the generally non-stationary nature of the TPCx-V workload, the load generated by various transactions and the corresponding workload remain relatively steady during the periods of interest. Consequently, our numerical investigation suggests that anomalies in the workload, focusing on a given time window, can be effectively detected using the bucket algorithm. In [64], it has been shown how to extend CUSUM to accommodate non-stationary workloads. Similarly, the adaptation of the bucket algorithm to handle non-stationary baselines, inspired by [64], is left as subject for future research.

Short-lived malicious jobs: As we showed in our results, the intensity and duration of the malicious activities can limit the applicability of our technique. For shorter bursts of attack, if the effect of the malicious activity does not interfere with the system's performance signature, the effectiveness of the approach is limited and a more computationally intensive mechanism may be required.

TABLE 11. Illustrating the feasibility of detecting anomalies through deviations of expected response time with respect to reference values. Large machines were observed during one week, and extra large machines were observed after several weeks of aging. Anomalies are identified when normalized distance values exceed 1.0. In the case of σ-normalized distance, anomalies are characterized by values surpassing 1.2. For large machines, σ-normalized distance is able to capture some anomalies that were not captured through normalized distance. For extra-large machines, both metrics work equivalently under the considered thresholds.

	response time				
machine number	average (s)	std (s)	max (s)	normalized distance	σ-normalized distance
20	55.670	43.260	320.250	1.34273999	1.56807735
21	25.290	15.073	152.064	0.96269509	1.19394199
22	27.880	17.780	222.970	1.01142753	1.25250309
23	30.759	39.680	866.810	1.06049061	1.44210709
24	25.270	15.680	324.350	0.96230008	1.20087977
25	27.870	25.430	555.000	1.01124819	1.32340161
26	30.250	16.725	355.000	1.05217391	1.26574604
27	30.800	51.020	949.000	1.06115418	1.50032089
28	30.026	51.200	925.000	1.04846707	1.49758472
29	24.600	14.900	137.000	0.94889103	1.18352060
30	25.990	15.110	142.000	0.97633358	1.20263350
31	25.090	14.300	342.000	0.95873137	1.18217287
32	25.090	14.300	148.000	0.95873137	1.18217287
33	23.400	12.800	125.700	0.92398815	1.14105595
34	24.788	13.600	121.000	0.95268842	1.16968829
35	27.160	16.070	167.000	0.99834589	1.22673099

(a) large machines

	response time				
machine number	average (s)	std (s)	max (s)	normalized distance	σ-normalized distance
36	27.42	18.98	239.00	1.00310957	1.26001358
37	27.86	20.05	289.00	1.01106877	1.27488026
38	22.10	12.59	164.00	0.89564336	1.12011624
39	27.38	18.20	229.00	1.00237964	1.25168200
40	21.80	13.10	187.00	0.88888889	1.12308930
41	22.70	13.40	204.00	0.90890891	1.13970008
42	24.80	15.20	176.00	0.95292988	1.18959108
43	24.60	15.40	255.00	0.94889103	1.18959108
44	22.20	12.60	167.00	0.89787664	1.12167607
45	24.70	15.08	216.00	0.95091434	1.18693122
46	30.20	20.09	235.00	1.05134900	1.29713696
47	23.20	14.10	205.00	0.91972250	1.15569326
48	24.20	14.40	164.00	0.94071914	1.17236143
49	21.20	11.50	137.00	0.87512900	1.09090909
50	24.86	15.03	186.00	0.95413548	1.18826333

(b) extra large machines

Fault model representativeness: The faults injected into the system are generated by stressing the underlying OS. In real systems, different attacks may affect multiple layers of the system stack. Nonetheless, as far as the subsumed OS states resulting from those attacks correspond to states generated by our fault injection, the considered failures are representative of those that occur in systems under operation [65].

Profile obsolescence: If user profiles are not stable, the number of false alerts can increase significantly, making the detection system useless until the next iteration. One possible approach is to segment time into windows and apply and tune the bucket algorithm at every time window. To determine the optimal window size, one may rely on techniques for learning in non-stationary environments [66].

Covert degradation: Single metrics may not suffice to detect anomalies. For instance, a detector using response time as its metric may miss-detect attacks that impair system availability if the few transactions that succeed in completing have their response time within the expected range. A similar effect occurs when measuring throughput in certain elastic systems. This limitation can be mitigated using multiple complementary metrics for anomaly detection.

VIII. CONCLUSION

In this work, we presented a methodology for anomaly detection based on performance degradation caused by security attacks for complex virtualized systems. The approach leverages an analytical model to optimize the detection technique in a principled way. Our experimental assessment indicates the method's effectiveness by injecting resource exhaustion attacks in a complex virtualized system. Results show that it is possible to detect anomalous behavior using the throughput of the business transactions with an average precision of 90% and recall of 86%. Our experimental results also bring awareness about the residual effects of high-intensity fault loads, which may persist for a significant time after the active attack has been interrupted. Such residual effects give rise to false positives, further motivating the use of analytical models to tune the false-positive rate in a principled fashion.

In future work, we intend to enhance the experimental evaluation by conducting a comprehensive campaign that addresses the challenges associated with adapting the CUSUM algorithm to complex environments like TPCx-V. Additionally, we plan to include the SPRT algorithm [52], [53] in our investigation to compare the advantages and dis-

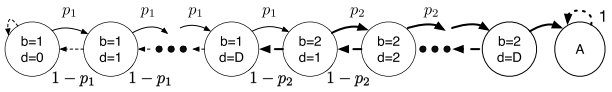


FIGURE 13. Bucket diagram for $B = 2$.

advantages of each approach. Furthermore, our objective is to expand the default configuration of TPCx-V by incorporating a larger system with more physical servers and diverse workload types, in order to assess the effectiveness of the proposed approach. We will focus on evaluating the microservice test application TeaStore [67], followed by exploring other distributed frameworks such as Apache Spark and Memcached.

APPENDIX A
BIRTH-DEATH PROCESS SUBSUMED BY THE BUCKET ALGORITHM

The models considered in our work are discrete time models, wherein transitions occur after a sample is collected. Nonetheless, for analytical purposes it is instrumental to also consider the corresponding continuous time models, wherein samples arrive according to a Poisson process, i.e., the mean time between samples is exponentially distributed. All the results derived in this appendix that are used throughout the rest of the paper hold for general distributions, as they ultimately rely on transition probabilities (transition rates, when used, appear to simplify presentation when leveraging results from M/M/1 and M/M/1/K queues, but the final results are a function of transition probabilities as opposed to transition rates).s

The bucket algorithm with a single bucket corresponds to an M/M/1 queue in discrete time. Let λ and μ be the birth and death rates, $\rho = \lambda/\mu$, and let $p = \mu/(\lambda + \mu)$ be the probability that a death (removal of ball from bucket) occurs before a birth (addition of ball into bucket). As mentioned in the above paragraph, all our results depend on λ and μ only through p, noting that[5]

$$\rho = \frac{1}{p} - 1.$$

Time is measured in number of collected samples, i.e., we consider a discrete time system where each time slot corresponds to the duration between two sample collections. The mean time to reach state N starting from state 0 is given by first passage time arguments (see Section 2.5.3 in [68]).

Let V_N be the mean time to reach state N from state 0, and let U_{n-1} be the mean time to reach state n from state $n-1$. Then,

$$V_N = \sum_{n=1}^{N} U_{n-1} \tag{29}$$

where

$$U_0 = \frac{1+\rho}{\rho} = \frac{1}{1-p} \tag{30}$$

[5]When considering a single bucket, we let $\rho = \rho_1$ and $p = p_1$.

$$U_n = 1 + p(U_{n-1} + U_n) = \frac{1 + pU_{n-1}}{1-p} \tag{31}$$

Solving the recursion above, we obtain an expression for U_n that we then use to express V_N in closed form.

A. DERIVATION OF U_n

1) DIRECT DERIVATION

Let $q = 1/(1-p)$ and $r = p/(1-p)$. Then,

$$U_0 = \frac{1}{1-p} \tag{32}$$

$$U_n = q\frac{1-r^n}{1-r} + r^n U_0 = q\frac{r^n - 1}{r-1} + r^n U_0 \tag{33}$$

Note that $1 - r = (1-2p)/(1-p)$. Then,

$$U_n = \frac{1-r^n}{1-2p} + r^n U_0 \tag{34}$$

Note also that

$$r = \frac{\mu}{\lambda} = \rho^{-1} = \frac{p}{1-p} \tag{35}$$

$$U_n = \frac{1-\rho^{-n}}{1-2p} + \rho^{-n}(1+\rho^{-1}) \tag{36}$$

$$= \frac{1+\rho}{\rho}\frac{1-\rho^{-n}}{1-\rho^{-1}} + \rho^{-n}(1+\rho^{-1}) \tag{37}$$

2) ALTERNATIVE DERIVATION

Next, we provide an alternative derivation for U_n, leveraging results about the M/M/1/K queue. As pointed out in the beginning of this appendix, the M/M/1/K model assumes that samples arrive according to a Poisson process, but this assumption is removed after uniformization, as detailed next.

The steady state probability of state $K+1$ at an M/M/1/K+1 system is given by

$$\tilde{\pi}_{K+1} = \rho^{K+1}\frac{1-\rho}{1-\rho^{K+2}} \tag{38}$$

Note also that

$$\frac{1}{\tilde{\pi}_{K+1}} = \frac{1}{\rho^{K+1}}\frac{1}{1-\rho} - \rho\frac{1}{1-\rho} \tag{39}$$

The mean time to go from state K to state $K+1$ in an M/M/1/K+1 system is

$$\tilde{U}_K = \frac{1}{\mu}\left(\frac{1}{\tilde{\pi}_{K+1}} - 1\right) \tag{40}$$

Now, note that the M/M/1/K+1 system is a continuous time system, whereas the system under consideration here is discrete time. We use uniformization to convert one into the other,

$$P = \frac{Q}{\lambda+\mu} + I \tag{41}$$

where P is the transition probability matrix of the discrete time system. Indeed, we let the uniformization rate equal $\lambda + \mu$, meaning that the uniformized system will make transitions on average every $1/(\lambda+\mu)$ time units, where time is measured

TABLE 12. Table of notation: a transition occurs after every sample. At state 0, we may have self-transitions.

Variable	Description
K	current bucket
d	number of items in current bucket
B	number of buckets
D	maximum depth of each bucket
V_N	mean number of transitions to reach state N from state 0
$V_{D,i,j}$	mean number of samples to increment the number of balls in bucket i by D units, starting from j balls at bucket i
U_n	mean number of transitions to reach state $n+1$ from state n
p	probability of sample response time being smaller than target, i.e., probability of a "good" sample, $p = \mu/(\lambda+\mu)$
ρ	mean number of "bad" samples collected until collecting a "good" one, i.e., until collecting one that reduces the number of balls in a bucket or maintains the state at 0, $\rho = (1/p) - 1 = \lambda/\mu$

according to the original continuous time system (for additional background on uniformization, see [69]). Therefore, the mean number of transitions to reach state $K+1$ from state K is given by (40) divided by $1/(\lambda+\mu)$,

$$U_K = \frac{\lambda+\mu}{\mu}\left(\frac{1}{\tilde{\pi}_{K+1}} - 1\right) = (\rho+1)\frac{\rho^{-K-1}-1}{1-\rho}. \quad (42)$$

Finally, (42) is equivalent to (37) replacing n by K.

B. DERIVATION OF V_N

Next, we derive an expression for V_N,

$$V_N = (\rho+1)\sum_{n=0}^{N-1}\frac{\rho^{-n-1}-1}{1-\rho} \quad (43)$$

$$= \frac{\rho+1}{(1-\rho)\rho^N}\left(\frac{1-\rho^N}{1-\rho} - \rho^N N\right) \quad (44)$$

In particular, if $N=1$ the above expression reduces to

$$V_1 = c = \frac{\rho+1}{1-\rho}\left(\frac{\rho^{-1}-1}{1-\rho} - 1\right) = \frac{1+\rho}{\rho} \quad (45)$$

as expected.

APPENDIX B
PROBABILITY OF FALSE POSITIVE BEFORE DETECTING AN ATTACK

Assuming that V_N can be roughly approximated by a constant, and that the mean time between attacks is exponentially distributed with mean α, the probability that we will get a false positive before we detect an attack is given by

$$f = e^{-V_N/\alpha} \quad (46)$$

Alternatively, if we approximate V_N by an exponential distribution,

$$f = \frac{1/V_N}{1/V_N + 1/\alpha} = \left(1 + \frac{V_N}{\alpha}\right)^{-1} \quad (47)$$

C. GENERAL CASE: VARYING NUMBER OF BUCKETS AND BUCKET DEPTH

Next, extend the above analysis for the case of multiple buckets. We focus on expectations (distributions are discussed in [70]). Recall that $A_B(D;(p_1,p_2,\ldots,p_B))$ is the mean time until absorption, measured in number of collected samples, accounting for B buckets of depth D each. Note that $A_B(D;(p_1,p_2,\ldots,p_B))$ is the mean time until a false alarm, starting from the initial state 0, and can be expressed either through $(p_1,p_2,\ldots,p_B)$ or $(\rho_1,\rho_2,\ldots,\rho_B)$,

$$A_B(D;(\rho_1,\rho_2,\ldots,\rho_B)) = V_{D+1,1,0} + \sum_{i=2}^{B} V_{D,i,1} \quad (48)$$

where $V_{D,i,j}$ is the mean number of samples to increment the number of balls in bucket i by D units, starting from the state wherein the system has j balls at bucket i. The expression of $V_{D+1,1,0}$ was previously computed, and is given by (44),

$$V_{D+1,1,0} = V_{D+1} \quad (49)$$

$$= \frac{\rho_1+1}{(1-\rho_1)\rho_1^{D+1}}\left(\frac{1-\rho_1^{D+1}}{1-\rho_1} - \rho_1^{D+1}(D+1)\right) \quad (50)$$

To derive an expression for $V_{D,i,1}$, we let $U_{j,i}$ be the mean time to increment the number of balls at bucket i by 1 unit, starting from $j+1$ balls.[6] Then,

$$V_{D,i,1} = \sum_{j=0}^{D-1} U_{j,i} \quad (51)$$

and

$$U_{j,i} = \begin{cases} U_{j+1}, & i=1, \quad j<D \\ 1 + p_i(U_{D-1,i-1} + U_{0,i}) = & \\ = (1+p_iU_{D-1,i-1})/(1-p_i), & i\geq 2, \quad j=0 \\ \rho_i^{-j}\left(\Delta_i + U_{0,i}\right) - \Delta_i, & i\geq 2, \quad j>0 \end{cases} \quad (52)$$

It follows that for $i \geq 2$,

$$V_{D,i,1} = \left(\frac{1+\rho_i}{1-\rho_i} + U_{0,i}\right)\left(\frac{1-\rho_i^{-D}}{1-\rho_i^{-1}}\right) - D\frac{1+\rho_i}{1-\rho_i}. \quad (53)$$

and, for $i=1$,

$$V_{D,1,1} = \sum_{j=0}^{D-1} U_{j+1} = V_{D+1} - V_1 \quad (54)$$

where V_D is given by (44).

APPENDIX C
DERIVATION OF METRICS OF INTEREST

D. SPECIAL CASE: $B=2$

In the particular where we have two buckets (Fig. 13),

$$A_2(D;\rho_1,\rho_2) = V_{D+1} + V_{D,2,1}$$

[6] Note that the dependence of $U_{j,i}$ on j occurs through the distinction bewteen cases $j=0$ and $j>0$.

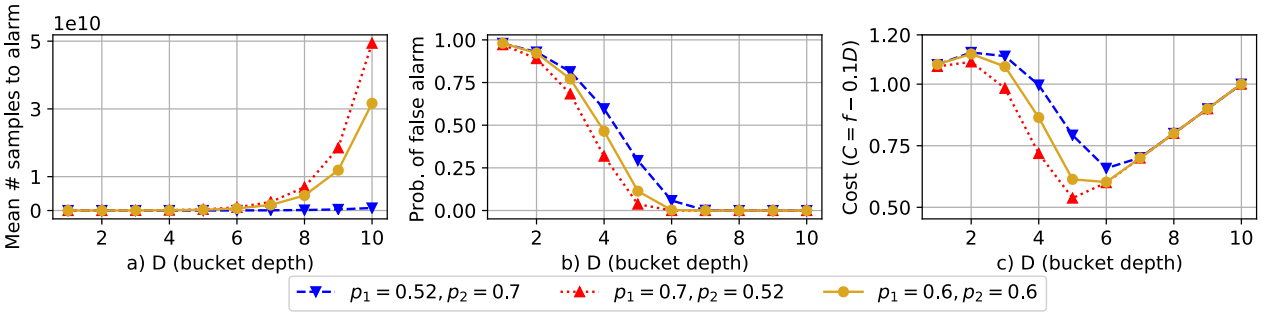


FIGURE 14. As the bucket depth increases, the probability of false alarm decreases but the time to detect attacks increases.

$$= \frac{\rho_1+1}{(1-\rho_1)\rho_1^{D+1}}\left(\frac{1-\rho_1^{D+1}}{1-\rho_1} - \rho_1^{D+1}(D+1)\right) + \left(\frac{1+\rho_2}{1-\rho_2} + U_{0,2}\right)\left(\frac{1-\rho_2^{-D}}{1-\rho_2^{-1}}\right) - D\frac{1+\rho_2}{1-\rho_2} \quad (55)$$

where

$$U_{0,2} = \frac{1+p_2U_{D-1,1}}{1-p_2} = \frac{1+p_2U_D}{1-p_2} \quad (56)$$

$$p_1 = \frac{1}{\rho_1+1}, \quad p_2 = \frac{1}{\rho_2+1} \quad (57)$$

$$U_D = (\rho_1+1)\frac{\rho_1^{-D-1}-1}{1-\rho_1} \quad (58)$$

Then, the expression of A_2 can be further simplified to

$A_2(D;\rho_1,\rho_2)$

$$= \frac{\rho_1+1}{(1-\rho_1)\rho_1^{D+1}}\left(\frac{1-\rho_1^{D+1}}{1-\rho_1} - \rho_1^{D+1}(D+1)\right) + \left(\frac{1+\rho_2}{1-\rho_2}\right)\left(1 + \frac{1-\rho_2^2+(1-\rho_2)U_D}{\rho_2(\rho_2+1)}\right)\left(\frac{1-\rho_2^{-D}}{1-\rho_2^{-1}}\right) - D\frac{1+\rho_2}{1-\rho_2}. \quad (59)$$

Similarly, A_2 can be expressed as a function of p_1, p_2 and D, as indicated in (3). It can be readily verified that (3) is equivalent to (59).

E. SPECIAL CASE: B = 2 AND D = 1

Let states 0, 1, 2 and F correspond to the initial state, 1 ball at bucket 1, 1 ball at bucket 2, and the final absorbing state, respectively. Next, we compute the mean number of samples to reach state F from state 0. It follows from (59) that

$$A_2(1;\rho_1,\rho_2) = \frac{1}{\pi_F} - 1 \quad (60)$$

where

$$\frac{1}{\pi_F} = \frac{1+\frac{1-p_1}{1-(1-p_1)p_2}+\frac{(1-p_1)^2}{1-(1-p_1)p_2}+\frac{(1-p_1)^2(1-p_2)}{1-(1-p_1)p_2}}{\frac{(1-p_1)^2(1-p_2)}{1-(1-p_1)p_2}}. \quad (61)$$

F. NUMERICAL EXAMPLES

Next, we illustrate the trade-off in the choice of the bucket depth. Figure 14(a) illustrates the behavior of (59). The red, yellow and blue lines correspond to three scenarios, respectively: 1) $p_1 = 0.7, p_2 = 0.52$; 2) $p_1 = 0.52, p_2 = 0.7$; 3) $p_1 = 0.6, p_2 = 0.6$. Recall that $1 - p_i$ is the probability of getting a "bad" sample at bucket i, that leads to an increase in the number of balls. As $1 - p_i$ increases, the mean time to alarm decreases. Figure 14(b) shows the probability of false alarm under the assumption of exponential time between attacks with rate $\alpha = 0.001$, and plots equation (8). Finally, Figure 14(c) shows that there is an optimal value of bucket depth that minimizes the cost, where cost is given by the difference between false alarm probability and normalized time to detect attacks, assumed to be proportional to the bucket depth.

G. ALARM DELAY EVALUATION

Next, we evaluate the alarm delay when the system is under attack and how it is impacted by: 1) parametrization of the detection algorithm and 2) the fault model. When the system is under attack, a lower bound on the number of samples until a true positive is given by L (Definition 2 on Section IV-B). Assuming all buckets are initially empty, we have $L = BD$. Additionally, since the resolution of our dataset is in seconds, there is a direct relationship between time to detection and the number of evaluated samples.

Next, we will conduct a detailed analysis of all the true positive alerts to gain insight into the behavior of the algorithm when faced with an attack. Our focus will be on assessing the timeliness of the detection, specifically by examining the data regarding the duration between the **start of the attack** and the **initial alert** generated by the **bucket algorithm**.

1) PARAMETRIZATION

Figure 15(a) shows the delay distribution for depth $D = 12$ and Figure 15(b) for $D = 15$. Those figures show data from all attack runs aggregated with all fault models, differing only by the bucket depth.

Figures 15(a) and 15(b) are quite similar. Nonetheless, for $D = 12$, we have more alerts (1007) than for $D = 15$ (825), noting that in this section we are accounting only for the first true positive alert. Additionally, recall that in our

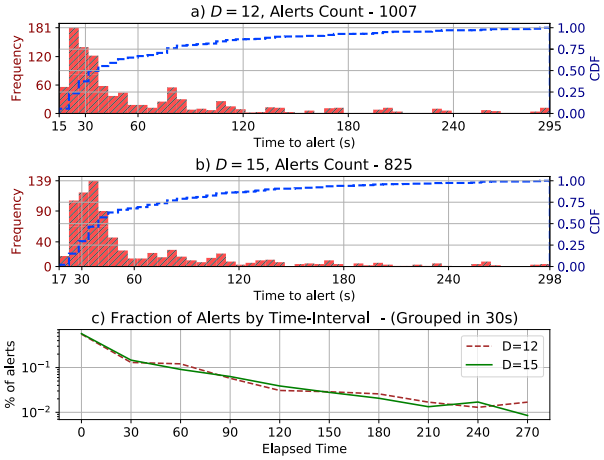


FIGURE 15. Time to detect attack, for $D = 12$ and $D = 15$.

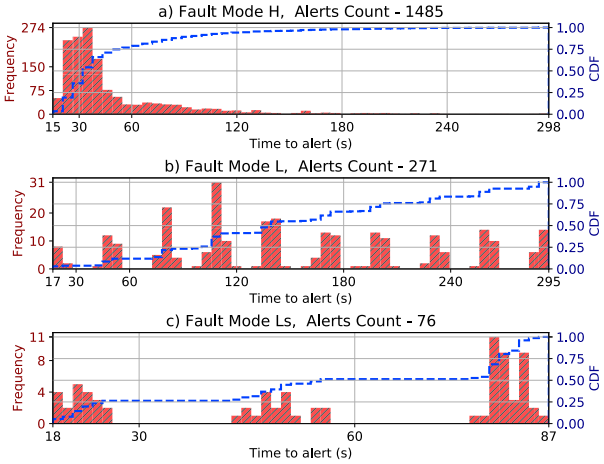


FIGURE 16. Time to detect attack by fault mode. Total of alerts = 1485.

methodology (Section III-C) an alarm is not necessarily generated for all alerts. In the extreme case where all transaction are impacted by an anomaly, in every subsystem, at similar times, we would get 4536 alerts for the scenario shown in Figures 15(a) and 15(b), corresponding to 2 attack phases, 3 fault models (Section V-A3), 21 runs (Section V-B3), 4 groups and 9 transactions (Section V-B1), i.e., $2 \times 3 \times 21 \times 4 \times 9 = 4536$.

Can a smaller bucket size in such an environment result in a higher number of alerts after the attack has begun? In addition, can the residual effects of the attack lead to an increased proportion of alerts towards transactions with $D = 12$ compared to the ones configured with $D = 15$? To address this question, we examined the fraction of alerts issued after the attack as a function of time for both $D = 12$ and $D = 15$. Figure 15(c) illustrates the fraction of alerts issued after the start of the attack, showing that there is no significant difference in the decay of alert proportions between $D = 12$ and $D = 15$. The figure suggests that the excess number of alerts generated when using a smaller bucket size ($D = 12$) follows a distribution similar to that of alerts produced with $D = 15$. In summary, our analysis suggests that the behavior of the algorithm after the attack begins is not very sensitive to D, leaving the time to detect an attack and the number of FPs as the key metrics to parametrize D.

2) FAULT MODEL

Next, we assess the impact of the various fault models on the transient behavior of the considered anomaly detection algorithm. During the assessment of various fault models, our focus is on examining the positive relationship between the intensity of the attack (including its frequency) and the number of alerts generated by the anomaly detection system.

Recall from Section V-A3 that our fault model stresses the system with a High intensity load (**H**) (for 300 seconds), with a Low intensity load (**L**) (10 periods of 15 seconds of stress, halting for 15 seconds) and Low intensity short load (**Ls**) (3 periods of 15 seconds of stress, halting for 15 seconds).

For the **H** fault mode (Figure 16(a)), the majority (75%) of the alerts were issued during the first minute after the attack occurred. For the **L** fault mode, we see a similar detection time in every attack performed in the complete interval (300s). The CDF of the number of alerts for the **L** fault model can be roughly approximated as a linear function of time, i.e, the number of alerts increases linearly over time. Finally, the **Ls** fault model (Figure 16(c)) comprises an observation interval of 90 seconds, and a significant number of alerts was raised in the last burst of attack. The last burst, in turn, shows the same behavior as the first 90 seconds of the **L** fault model.

To summarize, the findings presented in this appendix indicate that the number of alerts generated over time exhibits two key characteristics: 1) it aligns with the intensity of the attack, and 2) it tracks the progression of the attack. When comparing the results for bucket sizes $D = 12$ and $D = 15$, it is observed that the former produces a greater number of alerts than the latter, while still adhering to trends 1) and 2). Furthermore, when comparing different fault models, it is evident that the number of alerts consistently increases during the active phase of the attack and tends to stabilize once this period concludes.

H. SENSITIVITY ANALYSIS

Next, we consider the sensitivity of the cost with respect to the parameter of interest, D. To that aim, we take the derivative of the cost with respect to D. Under the deterministic model introduced in Section IV-D,

$$\frac{\partial C}{\partial D} = B + w\frac{\partial f_B(D)}{\partial D} = B - w\left(e^{-A_B(D)\alpha}\frac{\partial A_B(D)}{\partial D}\alpha\right) \quad (62)$$

Note that as D grows, the term multiplying w in the above expression vanishes. Indeed, the derivative of the cost tends to B as D grows to infinity, as for large enough D there will be virtually no false positives and the cost will be due to the time to detect anomalies when they in fact occur, i.e., time to detect true positives.

Figure 17 shows how the cost varies as a function of D, for $B = 2$ and $w = 20.646$ (see Section V-C). We let p_1 and p_2 equal 0.466 and 0.714, respectively. Note that the cost is robust against changes in D, i.e., it varies in a small range

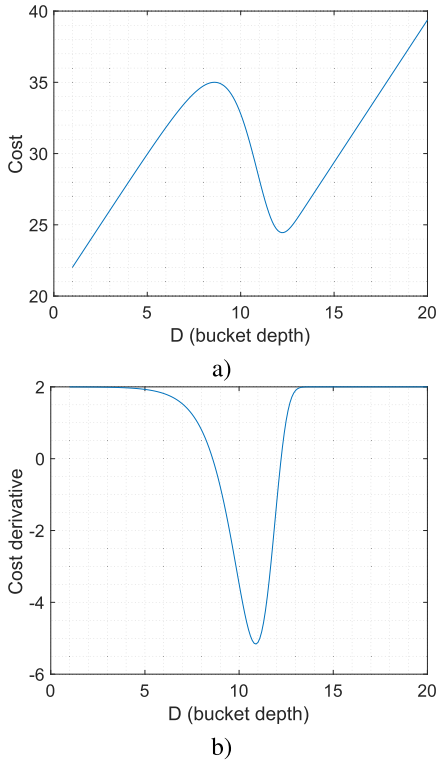


FIGURE 17. Sensitivity analysis when $w = 20.646$.

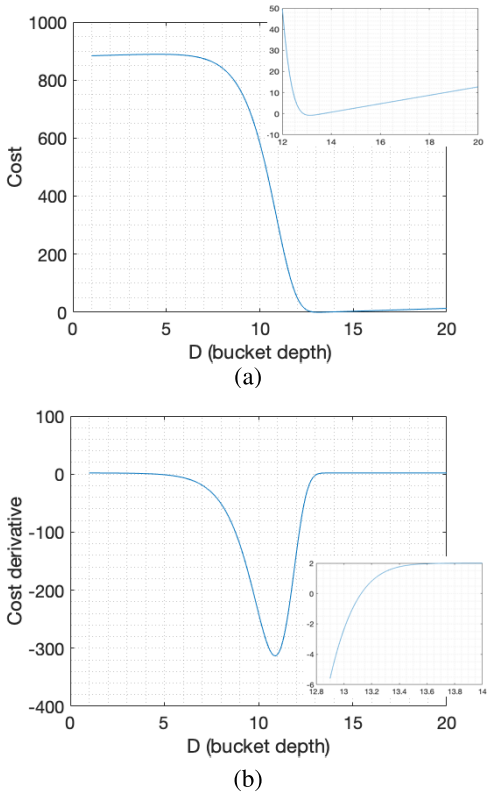


FIGURE 18. Sensitivity analysis when $w = 909$.

(Figure 17(a)). Similar observation holds for the example in Figure 14. The derivative of the cost also varies in a small range (Figure 17(b)). The cost has a local minimum around $D = 13$ and a global minimum at $D = 1$.

Next, we let $w = 909$ (Figure 18). Under this configuration, the cost function exhibits a single local minimum, once again centered around $D = 13$. However, now the cost is significantly more sensitive to variations in D. Based on these observations, we infer that if the objective of the soft constraint problem is to reliably capture the local minimum of the hard constraint problem, it is sufficient to consider smaller values for w. However, if it is desired for the Problem with Soft Constraints to possess a unique local minimum corresponding to the solution of the Problem with Hard Constraints, larger values of w may be necessary, albeit at the cost of reduced robustness to changes in D.

REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Dec. 2003.

[2] A. S. Tanenbaum, *Modern Operating Systems*, 3rd Ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2009. [Online]. Available: https://www.worldcat.org/oclc/254320777

[3] (2020). *83% Of Enterprise Workloads Will Be In The Cloud By 2020.* [Online]. Available: https://www.forbes.com/sites/louiscolumbus/2018/01/07/83-of-enterprise-workloads-will-be-in-the-cloud-by-2020

[4] Intel. (2019). *Unexpected Page Fault in Virtualized Environment Advisory*. [Online]. Available: https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00317.html

[5] DigitalOcean. (2019). *DigitalOcean Reply to Intel Security Advisory*. [Online]. Available: https://hup.hu/index.php/node/166970

[6] M. Wallschläger, A. Gulenko, F. Schmidt, O. Kao, and F. Liu, "Automated anomaly detection in virtualized services using deep packet inspection," *Proc. Comput. Sci.*, vol. 110, pp. 510–515, Jan. 2017.

[7] A. Gulenko, M. Wallschläger, F. Schmidt, O. Kao, and F. Liu, "Evaluating machine learning algorithms for anomaly detection in clouds," in *Proc. IEEE Int. Conf. Big Data*, Dec. 2016, pp. 2716–2721.

[8] I. Bojanova, J. Zhang, and J. Voas, "Cloud computing," *IT Prof.*, vol. 15, no. 2, pp. 12–14, Mar. 2013.

[9] M. Grottke, A. Avritzer, D. S. Menasché, L. P. de Aguiar, and E. Altman, "On the efficiency of sampling and countermeasures to critical-infrastructure-targeted malware campaigns," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 4, pp. 33–42, Feb. 2016.

[10] T. Zoppi, A. Ceccarelli, and A. Bondavalli, "Unsupervised algorithms to detect zero-day attacks: Strategy and application," *IEEE Access*, vol. 9, pp. 90603–90615, 2021.

[11] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Reference Format*, vol. 41, no. 15, pp. 1–58, 2009, doi: 10.1145/1541880.1541882.

[12] A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B. D. Payne, "Evaluating computer intrusion detection systems: A survey of common practices," *ACM Comput. Surv. (CSUR)*, vol. 48, no. 1, p. 12, 2015.

[13] A. Avritzer, A. Bondi, M. Grottke, K. S. Trivedi, and E. J. Weyuker, "Performance assurance via software rejuvenation: Monitoring, statistics and algorithms," in *Proc. Int. Conf. Dependable Syst. Netw. (DSN)*, 2006, pp. 435–444.

[14] A. Bond, D. Johnson, G. Kopczynski, and H. R. Taheri, "Architecture and performance characteristics of a PostgreSQL implementation of the TPC-E and TPC-V workloads," in *Proc. Technol. Conf. Perform. Eval. Benchmarking*, 2013, pp. 77–92.

[15] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell, "Fault injection for dependability validation: A methodology and some applications," *IEEE Trans. Softw. Eng.*, vol. 16, no. 2, pp. 166–182, Feb. 1990.

[16] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, and G. H. Leber, "Comparison of physical and software-implemented fault injection techniques," *IEEE Trans. Comput.*, vol. 52, no. 9, pp. 1115–1133, Sep. 2003.

[17] D. M. Nicol, W. H. Sanders, and K. S. Trivedi, "Model-based evaluation: From dependability to security," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 48–65, Jan. 2004.

[18] K. S. Trivedi, D. S. Kim, A. Roy, and D. Medhi, "Dependability and security models," in *Proc. 7th Int. Workshop Design Reliable Commun. Netw.*, Oct. 2009, pp. 11–20.

[19] S. Kumar, J. Turner, and J. Williams, "Advanced algorithms for fast and scalable deep packet inspection," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, Dec. 2006, pp. 81–92.

[20] C. F. Gonçalves, D. S. Menasché, A. Avritzer, N. Antunes, and M. Vieira, "A model-based approach to anomaly detection trading detection time and false alarm rate," in *Proc. Medit. Commun. Comput. Netw. Conf. (MedComNet)*, Jun. 2020, pp. 1–8.

[21] S. Garg, K. Kaur, N. Kumar, G. Kaddoum, A. Y. Zomaya, and R. Ranjan, "A hybrid deep learning-based model for anomaly detection in cloud datacenter networks," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 3, pp. 924–935, Sep. 2019, doi: 10.1109/TNSM.2019.2927886.

[22] A. Diamanti, J. M. S. Vilchez, and S. Secci, "LSTM-based radiography for anomaly detection in softwarized infrastructures," in *Proc. 32nd Int. Teletraffic Congr.*, Y. Jiang, H. Shimonishi, and K. Leibnitz, Eds. Osaka, Japan, Sep. 2020, pp. 28–36, doi: 10.1109/ITC3249928.2020.00012.

[23] C. Lee, J. Hong, D. Heo, and H. Choi, "Sequential deep learning architectures for anomaly detection in virtual network function chains," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Jeju Island, (South) Korea, Oct. 2021, pp. 1163–1168, doi: 10.1109/ICTC52510.2021.9621043.

[24] M. Rogers, P. Weigand, J. Happa, and K. Rasmussen, "Detecting CAN attacks on J1939 and NMEA 2000 networks," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 3, pp. 2406–2420, May/Jun. 2023.

[25] K. Doshi, Y. Yilmaz, and S. Uludag, "Timely detection and mitigation of stealthy DDoS attacks via IoT networks," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 5, pp. 2164–2176, Sep. 2021.

[26] P. Dash, G. Li, Z. Chen, M. Karimibiuki, and K. Pattabiraman, "PID-piper: Recovering robotic vehicles from physical attacks," in *Proc. 51st Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2021, pp. 26–38.

[27] T. Hunt, Z. Jia, V. Miller, C. J. Rossbach, and E. Witchel, "Isolation and beyond: Challenges for system security," in *Proc. Workshop Hot Topics Operating Syst.*, May 2019, pp. 96–104.

[28] S. Krishna and B. Rani, "Virtualization security issues and mitigations in cloud computing," in *Proc. 1st Int. Conf. Comput. Intell. Inform.* Cham, Switzerland: Springer, 2017, pp. 117–128.

[29] N. Gruschka and M. Jensen, "Attack surfaces: A taxonomy for attacks on cloud services," in *Proc. IEEE 3rd Int. Conf. Cloud Comput.*, Jul. 2010, pp. 276–279.

[30] S. S. Alarifi and S. D. Wolthusen, "Detecting anomalies in IaaS environments through virtual machine host system call analysis," in *Proc. Int. Conf. Internet Technol. Secured Trans.*, Dec. 2012, pp. 211–218.

[31] N. Antunes and M. Vieira, "On the metrics for benchmarking vulnerability detection tools," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Rio de Janeiro, Brazil, Jun. 2015, pp. 505–516.

[32] A. R. Ramtin, P. Nain, D. S. Menasche, D. Towsley, and E. D. S. E. Silva, "Fundamental scaling laws of covert DDoS attacks," *Perform. Eval.*, vol. 151, Nov. 2021, Art. no. 102236.

[33] J. Ho, M. Wright, and S. K. Das, "Fast detection of mobile replica node attacks in wireless sensor networks using sequential hypothesis testing," *IEEE Trans. Mobile Comput.*, vol. 10, no. 6, pp. 767–782, Jun. 2011.

[34] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Proc. IEEE Symp. Secur. Privacy*, May 2004, pp. 211–225.

[35] J. Díaz-Verdejo, J. Muñoz-Calle, A. E. Alonso, R. E. Alonso, and G. Madinabeitia, "On the detection capabilities of signature-based intrusion detection systems in the context of web attacks," *Appl. Sci.*, vol. 12, no. 2, p. 852, Jan. 2022.

[36] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni, "Automated anomaly detection and performance modeling of enterprise applications," *ACM Trans. Comput. Syst.*, vol. 27, no. 3, pp. 1–32, Nov. 2009.

[37] A. Avritzer, R. Tanikella, K. James, R. G. Cole, and E. Weyuker, "Monitoring for security intrusion using performance signatures," in *Proc. 1st Joint WOSP/SIPEW Int. Conf. Perform. Eng.*, Jan. 2010, pp. 93–104.

[38] T. Wang, J. Xu, W. Zhang, Z. Gu, and H. Zhong, "Self-adaptive cloud monitoring with online anomaly detection," *Future Gener. Comput. Syst.*, vol. 80, pp. 89–101, Mar. 2018, doi: 10.1016/j.future.2017.09.067.

[39] C. Rosenthal, *Principles of Chaos Engineering*. San Francisco, CA, USA: USENIX Association, Mar. 2017.

[40] C. Rosenthal and N. Jones, *Chaos Engineering: System Resiliency in Practice*. Springfield, MO, USA: O'Reilly, 2020.

[41] K. A. Torkura, M. I. H. Sukmana, F. Cheng, and C. Meinel, "CloudStrike: Chaos engineering for security and resiliency in cloud infrastructure," *IEEE Access*, vol. 8, pp. 123044–123060, 2020.

[42] A. Avritzer, A. Bondi, and E. J. Weyuker, "Ensuring stable performance for systems that degrade," in *Proc. 5th Int. Workshop Softw. Perform.*, Jul. 2005, pp. 43–51.

[43] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: High-bandwidth and reliable covert channel attacks inside the cloud," *IEEE/ACM Trans. Netw.*, vol. 23, no. 2, pp. 603–615, Apr. 2015.

[44] J. D. Musa, "Operational profiles in software-reliability engineering," *IEEE Softw.*, vol. 10, no. 2, pp. 14–32, Mar. 1993.

[45] J. Ericson, M. Mohammadian, and F. Santana, "Analysis of performance variability in public cloud computing," in *Proc. IEEE Int. Conf. Inf. Reuse Integr. (IRI)*, Aug. 2017, pp. 308–314.

[46] W. Lin, C. Xiong, W. Wu, F. Shi, K. Li, and M. Xu, "Performance interference of virtual machines: A survey," *ACM Comput. Surv.*, vol. 55, no. 12, pp. 1–37, Mar. 2023, doi: 10.1145/3573009.

[47] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 123–137, doi: 10.1145/2785956.2787472.

[48] M. Compastié, R. Badonnel, O. Festor, and R. He, "From virtualization security issues to cloud protection opportunities: An in-depth analysis of system virtualization models," *Comput. Secur.*, vol. 97, Oct. 2020, Art. no. 101905.

[49] R. Patil and C. Modi, "An exhaustive survey on security concerns and solutions at different components of virtualization," *ACM Comput. Surv.*, vol. 52, no. 1, pp. 1–38, Jan. 2020.

[50] D. Sgandurra and E. Lupu, "Evolution of attacks, threat models, and solutions for virtualized systems," *ACM Comput. Surveys*, vol. 48, no. 3, pp. 1–38, Feb. 2016.

[51] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, nos. 1–2, pp. 100–115, 1954.

[52] O. A. Grigg, V. T. Farewell, and D. J. Spiegelhalter, "Use of risk-adjusted CUSUM and RSPRTcharts for monitoring in medical contexts," *Stat. Methods Med. Res.*, vol. 12, no. 2, pp. 147–170, Apr. 2003.

[53] A. Wald, "Sequential tests of statistical hypotheses," *Ann. Math. Statist.*, vol. 16, no. 2, pp. 117–186, Jun. 1945.

[54] S.-F. Yang and S. W. Cheng, "A new non-parametric CUSUM mean chart," *Qual. Rel. Eng. Int.*, vol. 27, no. 7, pp. 867–875, Nov. 2011.

[55] *TPC EXPRESS BENCHMARK V (TPCx-V) Specification, Transaction Processing Performance Council (TPC)*, TPC, Apr. 2019.

[56] A. Bond, D. Johnson, G. Kopczynski, and H. R. Taheri, "Profiling the performance of virtualized databases with the TPCx-V benchmark," in *Proc. Technol. Conf. Perform. Eval. Benchmarking*, 2015, pp. 156–172.

[57] B. Groza and M. Minea, "Formal modelling and automatic detection of resource exhaustion attacks," in *Proc. 6th ACM Symp. Inf., Comput. Commun. Secur.*, Mar. 2011, pp. 326–333.

[58] Ubuntu. (2019). *Stress NG*. [Online]. Available: https://kernel.ubuntu.com/ cking/stress-ng/

[59] S. Ji, K. Ye, and C.-Z. Xu, "CMonitor: A monitoring and alarming platform for container-based clouds," in *Proc. Int. Conf. Cloud Comput.* Cham, Switzerland: Springer, 2019, pp. 324–339.

[60] M. Duarte, "detecta: A Python module to detect events in data," Tech. Rep., 2020, doi: 10.5281/zenodo.4598962.

[61] M. J. Zaki and J. W. Meira, *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2014.

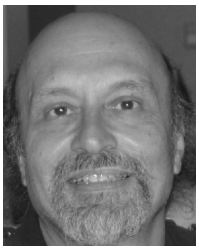
[62] Y. Gan, Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, and B. Jackson, "An open-source benchmark suite for microservices and their hardware–software implications for cloud & edge systems," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2019, pp. 3–18.

[63] A. Avritzer, R. Britto, C. Trubiani, M. Camilli, A. Janes, B. Russo, A. van Hoorn, R. Heinrich, M. Rapp, J. Henß, and R. K. Chalawadi, "Scalability testing automation using multivariate characterization and detection of software performance antipatterns," *J. Syst. Softw.*, vol. 193, Nov. 2022, Art. no. 111446, doi: 10.1016/j.jss.2022.111446.

[64] V. M. De Oca, D. R. Jeske, Q. Zhang, C. Rendon, and M. Marvasti, "A cusum change-point detection algorithm for non-stationary sequences with application to data network surveillance," *J. Syst. Softw.*, vol. 83, no. 7, pp. 1288–1297, Jul. 2010.
[65] J. Arlat, J.-C. Fabre, and M. Rodriguez, "Dependability of COTS microkernel-based systems," *IEEE Trans. Comput.*, vol. 51, no. 2, pp. 138–163, 2002.
[66] M. Sayed-Mouchaweh and E. Lughofer, *Learning in Non-Stationary Environments: Methods and Applications*. Berlin, Germany: Springer, 2012.
[67] J. von Kistowski, S. Eismann, N. Schmitt, A. Bauer, J. Grohmann, and S. Kounev, "TeaStore: A micro-service reference application for benchmarking, modeling and resource management research," in *Proc. IEEE 26th Int. Symp. Model., Anal., Simul. Comput. Telecommun. Syst. (MASCOTS)*, Sep. 2018, pp. 223–236.
[68] M. Zukerman, "Introduction to queueing theory and stochastic teletraffic models," 2013, *arXiv:1307.2968*.
[69] D. P. Heyman and M. J. Sobel, *Stochastic Models in operations Research. 1. Stochastic Processes and Operating Characteristics*. New York, NY, USA: McGraw-Hill, 1982.
[70] J. A. Fill, "The passage time distribution for a birth-and-death chain: Strong stationary duality gives a first stochastic proof," *J. Theor. Probab.*, vol. 22, no. 3, pp. 543–557, Sep. 2009.

CHARLES F. GONÇALVES received the B.Sc. and M.Sc. degrees in computer science from the Federal University of Minas Gerais (UFMG), Brazil. He is currently pursuing the Ph.D. degree in informatics engineering with the University of Coimbra (UC). He is a member of the Software and System Engineering (SSE) Group, Centre for Informatics and Systems of the University of Coimbra (CISUC), and the SPEC Research Security Benchmarking Working Group. His research interests include virtualization security, security benchmarking, data processing, and software development.

DANIEL SADOC MENASCHÉ (Member, IEEE) received the Ph.D. degree in computer science from the University of Massachusetts Amherst, in 2011. He is currently an Associate Professor with the Department of Computer Science, Federal University of Rio de Janeiro, Brazil. His research interests include modeling, analysis, security, and the performance evaluation of computer systems. He was a recipient of the best paper awards from GLOBECOM 2007, CoNEXT 2009, INFOCOM 2013, and ICGSE 2015. He is an Alumni Affiliated Member of the Brazilian Academy of Sciences.



ALBERTO AVRITZER (Member, IEEE) received the B.Sc. degree in computer engineering from the Technion—Israel Institute of Technology, the M.Sc. degree in computer science from the Federal University of Minas Gerais, Brazil, and the Ph.D. degree in computer science from the University of California, Los Angeles. He is currently the Founder and the CEO of EsulabSolutions Inc., the preferred source for methods for scalability assessment of mission-critical systems in the telecom and banking domains. He led the automated performance testing and analysis of New York subway public announcement and customer information system (PA/CIS), where he was responsible for the certification of system scalability to support 550 stations. He was a Lead Performance Engineer with Sonatype, Fulton, MD, USA, and a Senior Member of the Technical Staff with the Department of Software Engineering, Siemens Corporate Research Inc., Princeton, NJ, USA. Before moving to Siemens Corporate Research Inc., he spent 13 years with AT&T Bell Laboratories, where he developed tools and techniques for performance testing and analysis. He spent the Summer of 1987 with IBM Research, Yorktown Heights. His research interests include software engineering, particularly software testing, monitoring, and rejuvenation of smoothly degrading systems, and metrics to assess software architecture. He has published more than 70 papers in journals and refereed conference proceedings in those areas. He is a Senior Member of ACM. He has presented several tutorials at international conferences, such as LADC, IEEE ISSRE, and ACM ICPE.

NUNO ANTUNES (Member, IEEE) received the Ph.D. degree from the University of Coimbra (UC), in 2014. Since 2008, he has been a Researcher in software security and dependability with the Centre for Informatics and Systems of the University of Coimbra (CISUC). He is currently an Assistant Professor with the Department of Informatics and Engineering, UC. His research interests include testing, fault injection, vulnerability injection, and benchmarking, which are applied to the assessment of the dependability and security of intelligent systems, virtualized environments, intrusion detection systems, web services, web and mobile applications, and data management systems.

MARCO VIEIRA (Member, IEEE) is currently a Full Professor with the University of Coimbra, Portugal. His research interests include dependability and security assessment and benchmarking, fault injection, software processes, and software quality assurance, subjects in which he has authored or coauthored more than 200 papers in refereed conferences and journals. He has participated and coordinated several research projects, both at the national and European level. He is the Vice Chair of the IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance and an Associate Editor of the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING journal. He has served as the program chair. He is on the program committees of the major conferences of the dependability area.