Detecting Anomalous Behavior of Black-Box Services Modeled with Distance-Based Online Clustering

Anton Gulenko*, Florian Schmidt*, Alexander Acker*, Marcel Wallschläger*, Feng Liu† and Odej Kao*
*Complex and Distributed IT-Systems Group, TU Berlin, Berlin, Germany
Email: {firstname.lastname}@tu-berlin.de
†Huawei European Research Center, Huawei Technologies, Munich, Germany
Email: feng.liu1@huawei.com

***Abstract*—Reliable deployment of services is especially challenging in virtualized infrastructures, where the deep technological stack and the multitude of components necessitate automatic anomaly detection and remediation mechanisms. Traditional monitoring services observe the system and generate alarms when the collected metrics exceed predefined thresholds. The fixed thresholds rely on expert knowledge and can lead to numerous false alarms, while abnormal behavior that spans over multiple metrics, components, or system layers, may not be detected.**

We propose to rectify these shortcomings by combining metrics from multiple infrastructure layers and analyzing them using data mining algorithms. In particular, we use an unsupervised online clustering algorithm to create a model of the normal behavior of each monitored component with minimal human interaction and no impact on the monitored system. When an anomaly is detected, a human administrator or automatic remediation system can subsequently revert the component into a normal state. An experimental evaluation resulted in a high accuracy of our approach, which shows that it is suitable for anomaly detection in productive systems.

***Keywords*-cloud computing; service virtualization; anomaly detection; machine learning**

I. Introduction

In times of cloud computing, outages of components happen frequently and must be anticipated and handled. Large services often make use of the elasticity provided by Infrastructure as a Service (IaaS) platforms and scale in, scale out and migrate automatically between different physical or virtual machines. On the one hand, this brings many advantages to businesses and service providers, but on the other hand it introduces in-transparencies between system layers and leads to a more fragile technological stack. Furthermore, the sheer number of components and subsystems in a large-scale cloud environment leads to a high fault rate.Faults must be hidden from clients to provide a reliable service, which is crucial to fulfill service quality expectations. This is exacerbated in scenarios like Network Function Virtualization (NFV) where customers have come to expect an especially high degree of service availability and service quality.

Service quality can suffer from anomaly situations that include a wider range of scenarios than complete outages of components. Generally speaking, an anomaly is a deviation from normal system behavior. The changed behavior can degrade the service quality without making the service, or one of its components, entirely unavailable. Anomalies often precede component outages, thus making them a good target for fault prediction. In other words, early remediation of an anomaly can prevent a more severe fault situation.

It is very time consuming and often impossible for human administrators to manually locate, understand, and handle every anomaly. Automatic mechanisms are inevitable when guaranteeing the availability of hundreds of services running on thousands of physical machines in multiple data centers. Traditional monitoring services observe various parts of the system and trigger alarms when manually defined thresholds of certain metrics are exceeded. Such alarms can trigger automatic actions like scaling out or helping administrators to identify the anomaly and localize it in the systems. However, they have two important drawbacks:

1) Every layer and component is monitored and analyzed individually.
2) Fixed thresholds require expert knowledge and must be continuously updated.

For big data centers this can lead to continuously large amounts of false alarms. Post-aggregation of alarms reduces the number of alarms handled by human administrators, but does not solve the underlying problem.

Our approach to monitoring and anomaly detection avoids these problems by acknowledging the multi-level nature of clouds and by taking advantage of modern data mining methods. Deep, cross-layer monitoring delivers metrics from the entire cloud stack - from the physical layer up to the service layer. Data mining techniques can automatically uncover hidden dependencies in the collected data that would otherwise be missed by humans. As a result, false alarms are reduced by not relying solely on thresholds of individual metrics, but additionally analyzing the correlations between metrics on all system layers. Furthermore, said correlations uncover new anomalies, that would otherwise remain undetected.

The anomaly detection step is realized by training a *normal behavior model* with minimal human interaction

and without labeled training data. The online clustering algorithm BIRCH [1] can create a memory-efficient summary of a continuous high-dimensional data stream. We leverage an adapted version of the BIRCH algorithm to create an abstraction of the data collected while a component is operating normally. After this initial training phase, the monitored system state can be compared with the resulting model to derive whether a host is operating normally or not.

We envision a closed loop self-healing cloud system where the anomaly detection is the first step, followed by a subsequent root cause analysis (RCA) and finally an automatic selection and execution of an appropriate remediation action to revert the system to a normal state [2].

An experimental evaluation shows that the proposed approach delivers high precision and recall measurements and is suitable for productive environments.

The core contributions of this work include:

1) An architectural and algorithmic approach for detecting and localizing anomalies in virtualized services.
2) Prototypical implementation and experimental evaluation of the proposed approach.

The rest of this paper is organized as follows. Section II gives an overview over related efforts in the field of anomaly detection for cloud computing platforms. Section III introduces terminology and describes our algorithmic approach in detail. Section IV describes a prototypical implementation of our approach, the corresponding evaluation testbed and results. Finally, section V concludes.

II. Related Work

Detecting anomalous behavior of virtualized services has been recognized as an important part of the efficient service management, leading to numerous related efforts in this field.

Various data-driven approaches have been proposed and evaluated for detecting anomalies in specific domains such as scale-out storage systems [3, 4], or NFV environments [5, 6, 7]. Such approaches rely on domain specific data, which potentially increases the accuracy of data mining algorithms, but on the other hand makes them less applicable to virtualized environments in general.

Less restrictive approaches are based on the analysis of system level monitoring data obtained from virtual machine hypervisors or directly from guest or hypervisor operating systems. Various supervised learning algorithms have been explored for the use of online anomaly detection [6, 8, 9]. Such classification algorithms rely on labeled training data to distinguish normal from abnormal behavior. Their practicality in productive systems is limited, since obtaining labeled training data necessitates the injection of artificial anomalies, which is usually not acceptable due to the impact on the service quality experienced by customers. This limitation exacerbated in virtualized environments, where the workload changes frequently, resulting in the need to periodically re-train the classification models and repeat the required anomaly injections.

Other approaches make use of statistical analyses of individual performance metric time series [10, 7, 11, 12] to detect deviations from usual resource consumption. While such analyses can be carried out online and without labeled data, the focus on individual performance metrics limits the scope of the analysis: anomalies that manifest in the relationship of multiple performance metrics can not be detected.

Finally, a number of related approaches depends on a service quality metric to distinguish normal from abnormal service behavior during runtime [13, 5, 14]. The predominant technique for clustering such labeled data are Self-Organizing Maps [15] and flagging each neuron as either normal or abnormal behavior. Such approaches target anomaly prediction rather then detection, potentially allowing a timely remediation before the anomaly occurs. On the other hand, relying on a service quality metric, which might not always be available, reduces the generality of the approach.

III. Modeling Normal Service Behavior

We consider a component anomalous when its behavior deviates from normal behavior. The behavior of a component C at time t is defined as a vector $M_{t,C}$ of n metrics $x_i \in \mathbb{R}$, where $i = 1, 2, ..., n$. Behavior metrics for C can also be values $x \notin \mathbb{R}$, e.g. categorical variables, in which case a mapping function is required to project such values into $\mathbb{R}$. The normal behavior of C is modeled through a function $normal_C : \mathbb{R}^n \rightarrow \{0, 1\}$, which returns 1, if the input vector describes normal behavior of C, and 0 otherwise.

The normal behavior model $normal_C$ can be realized through a number of binary classification models. However, only few models fulfill the following requirements of an anomaly detection component for a self-healing cloud infrastructure:

1) It should work online and with limited memory.
2) It should have a low detection latency.
3) It should have minimal interference with the monitored system.
4) It should not depend on training data from anomalous behavior.

To meet these requirements, we propose to realize $normal_C$ by dividing the metric space R^n into two distinct sub spaces for normal and abnormal behavior. The subspace for normal behavior of C is described by a set $\mathcal{N}_C$ of centroids (c, r), where $c \in \mathbb{R}^n$ is the center and $r \in \mathbb{R}$ is the radius of the centroid. The subspace for the abnormal behavior is the metric space not occupied by any centroid, i.e. $\mathbb{R}^n \setminus \bigcup \mathcal{N}_C$. Therefore, the normal behavior model is

defined as:

$$normal_C(M) = \begin{cases} 1 & \text{iff } \exists (c,r) \in \mathcal{N}_C \;\middle|\; ||M-c|| \leq r \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

In other words, $normal_C$ returns a positive result, iff any of the centroids contains the incoming sample.

The problem is now reduced to creating the centroid set $\mathcal{N}_C$. For that purpose, a number of normal behavior measurements $\mathcal{B}_C^{norm}$ must be collected from the observed component C. Note that no anomaly data is necessary. The normal behavior data can be obtained by simply observing the component for a certain period of time, following the principle that each component behaves normally the majority of time. In a system with frequent anomalous behavior, an administrator can assert the correct operation for a given observation period.

We construct the centroid set $\mathcal{N}_C$ through the first phase of the online clustering algorithm BIRCH[1]. In that phase, BIRCH creates a micro-cluster approximation of a continuous input stream of vector data. The complete BIRCH algorithm consists of three additional phases: condensing the micro-cluster representation to a smaller memory footprint, an additional global clustering step using the k-means algorithm, and an optional offline refinement step of the resulting clusters. We omit those additional phases and use the micro-clusters produced in the first phase as the centroid set $\mathcal{N}_C$ describing the normal operation of component C.

BIRCH operates iteratively and does not store the raw input data, which fulfills the previously stated requirements. The micro-clusters are maintained as *cluster features (CFs)*, where each CF summarizes a subset of previously observed input vectors P by storing the number of vectors, the sum, and the squared sum of the vectors. In other words, a CF summarizing a vector set P is defined as the following tuple:

$$(|P|, \sum_{x \in P} x, \sum_{x \in P} x^T x)$$

The stored data can be updated iteratively, and the center and radius of the summarized centroid can be computed from it:

$$c_P = \frac{\sum\limits_{x \in P} x}{|P|}$$

$$r_P = \sqrt{\frac{|\sum\limits_{x \in P} x^T x|}{|P|} - \left(\frac{|\sum\limits_{x \in P} x|}{|P|} \right)^2}$$

Therefore, the centroid set $\mathcal{N}_C$ necessary to compute $normal_C$ can be obtained at any time. To optimize the insertion of new input data, the CFs are maintained in a height-balanced tree.

Besides meeting the previously stated requirements, BIRCH also allows online refinements of the normal operation model, as additional data vectors can be inserted into the CF tree at any time. This allows the model to adapt to changed system behavior, like changed workloads. Model updates can be triggered externally, either automatically when the system workload is changed, or manually by an administrator.

A. Obtaining Behavioral Data

Various types of components can be analyzed with the proposed anomaly detection approach. To achieve a high coverage of anomaly incidents, all layers of the cloud infrastructure should be monitored, including the physical network, physical hosts, virtual hosts, and virtualized services. The number of metrics collected for each component can vary, since an individual normal behavior model is maintained for each component.

The core requirements for the data collection are a high sampling frequency while causing a low resource overhead for the monitored system. In general, the sampling frequency should be as high as possible while staying below a given resource usage limit. A CPU and network resource overhead of 1-2% is often acceptable, which can be achieved with a sampling interval of a few hundred milliseconds.

Data for physical and virtual hosts can be obtained from the operating system. Depending on the cloud infrastructure, an intrusive data collection for virtual machines might not be acceptable, in which case the hypervisor can provide similar performance metrics. Separate data can also be collected for individual service processes. This allows monitoring individual services sharing the same virtual machines, but requires a data collection running directly on the VM. Certain high level network metrics like throughput, latency and packet drops can also be collected from most operating systems. For more detailed network anomaly detection, additional network specific monitoring protocols like SNMP[1], or newer monitoring techniques like In-Band Network Telemetry[2] can be used.

The data sources described so far are completely service agnostic, and enable anomaly detection for black-box services. Additional service specific metrics can increase the anomaly detection accuracy, but reduce the portability of the implementation. Potentially interesting metrics include service layer load (e.g. handled requests per second) or the quality of service.

IV. Evaluation

This section presents an evaluation of the proposed anomaly detection approach. First, a description of the prototypical implementation highlights various design decisions of the implementation. The evaluation testbed and results are presented afterwards.

[1] https://tools.ietf.org/html/rfc1157

[2] https://p4.org/assets/INT-current-spec.pdf (accessed March 2018)

A. Prototype Architecture

The architecture of the evaluation prototype follows a previously proposed design [2] and is summarized in figure 1. The scope of the anomaly detection covers the behavior of physical and virtual hosts. The following high-level components are part of the evaluation prototype:

- OpenStack private cloud platform
- Video-on-demand service workload (load-generation clients, load-balancers, RTMP servers)
- Data collection
- Model building and data analysis
- Anomaly injection for evaluation purposes

The monitored cloud infrastructure consists of an OpenStack private cloud, hosting a load-balanced and replicated video-on-demand service. The data collection, model building, and online data analysis are implemented locally on every host in the setup. This simulates a private cloud environment with full access to the running VMs, e.g. a communication service provider. A centralized aggregation service collects all intermediate data analysis results, provides a visualization and logs the data for later evaluation. In order to show that the anomaly detection is able to handle unforeseen anomalies, an anomaly injection component is running on every host.

B. Data Collection

The data collection is implemented as a service with elevated privileges, which samples local resource usage data and provides various metrics in the form of time series. While running locally on every host, it can store the collected data to log files or send them over the network for the purpose of visualization or further analysis. The Linux kernel provides resource usage information through the `/proc` filesystem, which is the main data source for the data collection service. The frequency of sampling data can be arbitrarily high and the data collection service is designed to minimize resource overhead induced by the sampling routine itself. While collecting data for the given evaluation, the sampling interval was configure to 250 milliseconds.

Each hypervisor host provides additional interfaces for sampling data about hosted virtualized resources. The data collection service implements data collection from the virtualization Application Programming Interface (API) Libvirt [16] and the OVSDB [17] protocol provided by the software switch Open vSwitch [18]. These APIs provide additional metrics about virtual machines and virtual network interfaces. Table I gives an overview over the metrics available from the different data sources.

C. Model Building and Data Analysis

The WEKA library [19], written in the Java programming language, implements many widely used data mining and machine learning algorithms, including the BIRCH clustering algorithm. The MOA library [20] is closely related to WEKA and provides streaming capable versions of many algorithms. The prototypical data analysis service consists of a modular wrapper around the MOA and WEKA libraries. It receives data streams from the data collection service, pushes the data through a pipeline of preprocessing steps and algorithms, and forwards the analysis results for further evaluation. In particular, the analysis pipeline consists of the following steps:

1) Data Normalization: The normalization of incoming data points is the main preprocessing step. As the clustering algorithm used for anomaly detection is based on the euclidean distance, it is rather sensitive to difference value ranges and distributions of input data. We perform a *standard score* normalization on every input value:

$$x_{out} = \frac{x_{in} - \mu}{\sigma}$$

Standard score normalization transforms each input metric to have a mean of 0 and a standard deviation of 1. The required values μ and σ were obtained by observing the monitored components during normal operation.

2) BIRCH Micro-Cluster Training: Since BIRCH needs no offline training step, the samples received at runtime are directly added into the CF tree. This step is only executed for input samples that have been marked as normal behavior samples. For this purpose the data collection service supports an API for attaching meta data tags to samples it produces. Depending on the strategy of collecting normal behavior data, an administrator can initialize a training phase when normal system operation is guaranteed.

3) Evaluation of the Normal Behavior Model: When the micro-clusters created by BIRCH sufficiently represent the host normal behavior, the resulting cluster model can be used to classify incoming samples as normal or abnormal. This processing step adds a meta data tag to the sample containing the ID of the cluster it is contained in. A missing cluster ID identifies a sample as a detected anomaly.

4) Post Processing: The post processing step attaches additional meta data to the outgoing samples. This includes information about the BIRCH clusterer and the processing time. The meta data is mainly used for evaluation purposes and would be disabled in a productive environment.

D. Evaluation Scenario

A high available OpenStack private cloud platform [21] forms the foundation of the evaluation scenario. The platform runs on 14 commodity servers with the following specifications:

- Intel Xeon X3450 (4 cores @ 2.66GHz)
- 16G RAM
- 3 x 1TB disk
- 2 x 1GBit Ethernet

The OpenStack controller is replicated on three physical machines, and there are two replicated network hosts. The

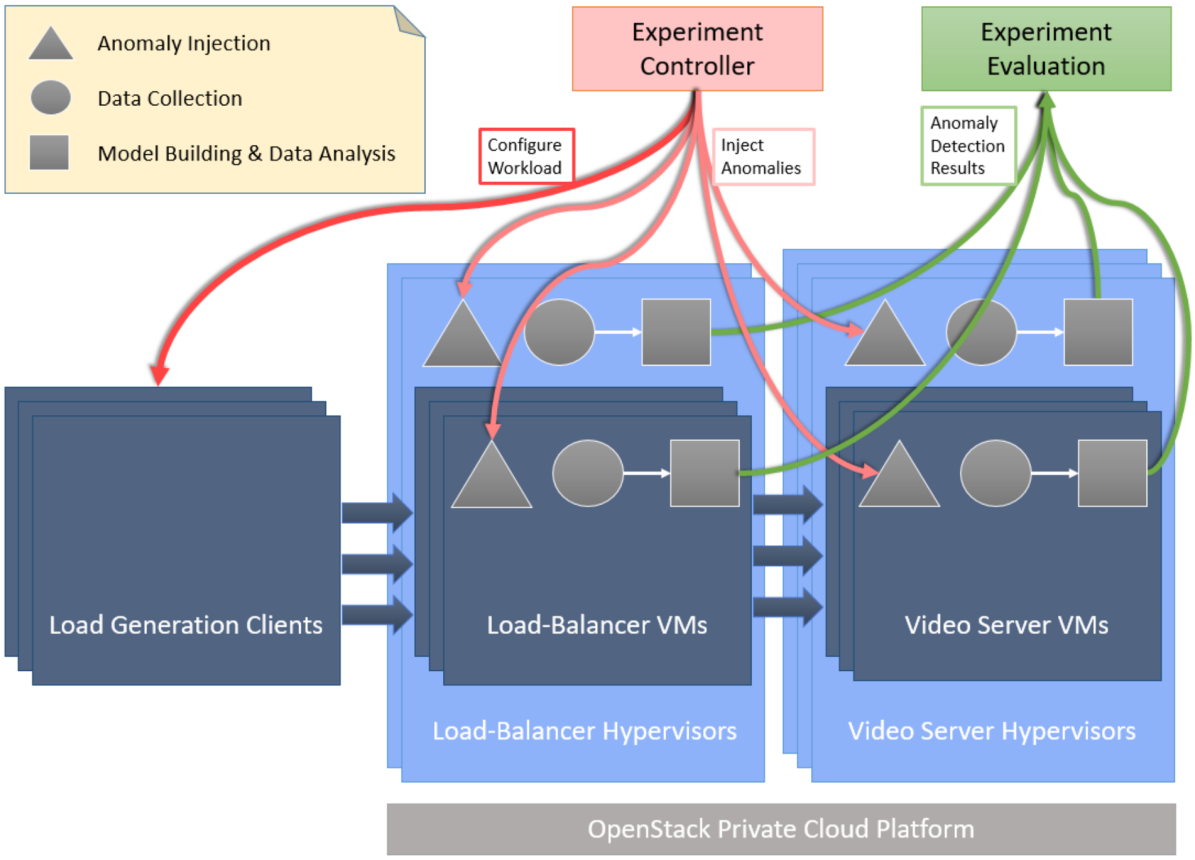

Figure 1. Prototype Architecture Overview

Metric	Description	Host-wide	Libvirt: Per VM	OVSDB: Per bridge
CPU	CPU utilization	×	×	
Load	The Linux kernel Load level			
RAM	RAM utilization	×	×	
Disk IO	Hard disk access in bytes and access times	×	×	
Disk space usage	Disk space per partition	×	×	
Network IO	Network utilization in bytes and packets	×	×	×
Network protocol counters	Protocol specific counters for IP, UDP, TCP and ICMP	×		
Number of processes		×		

Table I
SYSTEM RESOURCES SAMPLED BY THE DATA COLLECTION SERVICE

KVM virtualization engine and the Open vSwitch virtual switch manage the virtual resources. The remaining nine physical machines are dedicated compute hosts.

A virtualized video-on-demand service runs on the cloud platform. Five of the nine compute hosts are dedicated to running twenty Virtual Machines (VMs) with a video streaming service. The service streams RTMP video content and is implemented using the RTMP module[3] of the Nginx web server[4]. Every service replica has a local copy of the streamed video content. Two additional compute nodes are hosting a total of eight load balancer VMs that distribute incoming client requests to the backend video servers in a round robin fashion. The remaining two compute nodes are used for load generation clients streaming available video content using the ffmpeg[5] command line tool. When starting

[3]https://github.com/arut/nginx-rtmp-module

[4]https://nginx.org/

[5]https://www.ffmpeg.org/

a video download, the clients select one of the load balancer hosts through round robin based DNS load balancing.

E. Evaluation Procedure and Anomaly Injection

To evaluate our anomaly detection approach, the presented testbed has been exposed to varying, random anomaly injections over a period of 45 hours. At the same time, the load generation clients were putting the video backends and load balancers under varying load. The load was adjusted randomly between 200 and 400 simultaneous video streams in regular intervals, causing medium to high load on the servers, without reducing the stream quality due to overload.

Every anomaly was injected for five minutes, with one minute of cooldown time between anomalies, resulting in 450 total anomaly events over the 45 hour evaluation period. After five minutes, the currently injected anomaly was automatically recovered. The anomalies were evenly distributed between the four main components of the evaluation testbed: physical and virtual load balancer hosts, and physical and virtual video backend hosts. Within each host group, the anomaly target host was chosen randomly. Every anomaly was randomly parameterized within certain bounds to vary the anomaly intensity. What follows is a description of the types of injected anomalies and their parameters. Table II shows the parameter value ranges defined for the different host and anomaly type combinations.

1) CPU Stress: The `stress-ng` tool[6] is used to consume CPU resources by launching threads spinning on redundant mathematical operations.

2) Memory Leak: A process continuously allocates and never releases additional memory. To ensure that the memory is actually physically allocated and not swapped out, the allocated memory regions are invalidated regularly. The increased memory consumption still quickly leads to high swap usage and CPU overhead.

3) Increased Network Latency: This anomaly utilizes the iptables mechanism[7] to increase the latency for each incoming and outgoing packet by a predefined time.

4) Packet Loss: Multiple network related anomalies leverage the `tc` tool (short for "traffic control") for configuring the network scheduler in the Linux kernel of the injected host. For the Packet Loss anomaly, `tc` configures the kernel continuously drop a give percentage of all packets going through the network interfaces of the injected host.

5) Decreased Network Bandwidth: The `tc` tool reduces the maximal throughput of all network interfaces of the injected host.

6) Packet Duplication: The `tc` tool configures the Linux kernel to randomly duplicate outgoing packets, thereby disturbing running TCP connections.

[6]http://kernel.ubuntu.com/~cking/stress-ng/

[7]https://www.netfilter.org/projects/iptables/index.html

7) HDD Stress: The `stress-ng` tool spawns a number of worker threads utilizing the hard disk by reading and writing temporary files. This anomaly is only injected on the video server VMs, as none of the other host groups would be affected by it.

8) Network Stress: The network access of the injected host is congested by downloading large files from a dedicated external HTTP server. The downloaded file is not written to disk to ensure that mainly the network is affected by this anomaly.

F. Evaluation Results

For every anomaly event, the detection success was recorded, and in case of success, the anomaly detection latency.

Table III reports the anomaly detection rate and latency per anomaly type. While most anomalies are detected perfectly, the memory leak and CPU stress anomalies have a lower detection rate on physical hosts. This is likely due to the bigger resource usage fluctuation on physical host, since they are exposed to the combined resource usage of all hosted VMs. The detection latency shows a similar pattern: while anomalies on virtual hosts are mostly detected within less than a second, anomalies on physical hosts take longer to manifest themselves enough trigger a detection. The memory leak anomaly is noteworthy, as it is subject to a longer detection latency both on physical and virtual hosts. This anomaly is the only evaluated anomaly that progresses over time and is only detected when a critical memory consumption is reached.

V. Conclusion

This paper presents an algorithmic approach to anomaly detection in a virtualized service infrastructures. Through deep, cross-layer data monitoring the top-to-bottom behavior of the technology stack can be modeled and analyzed. The normal behavior of each system component is modeled through a tree of micro-clusters maintained by the BIRCH algorithm. At runtime, the resulting models are able to differentiate between normal and abnormal component behavior. This approach overcomes shortcomings of traditional monitoring services by creating the normal behavior model automatically and taking advantage of a holistic view of the system. An experimental evaluation showed that the proposed approach results in a high detection rate and low detection latency, while maintaining a low false alarm rate. However, the proposed algorithm alone is not able to guarantee the continuous operation of virtualized services. Interesting future efforts in this direction include the automatic selection and execution of recovery routines, measurement of their success through a feedback loop, and a multi-level aggregation of analysis results to support arbitrarily large infrastructures.

Anomaly	Load-Balancer Hypervisors	Backend Hypervisors	Load-Balancer VMs	Backend VMs
CPU Stress (all cores)	90-100%	90-100%	90-100%	90-100%
Memory Leak	2.5-500 MB/s	2.5-500 MB/s	1-200 MB/s	1-200 MB/s
Packet Loss	5-15 %	5-15 %	5-15 %	5-15 %
Packet Duplication	5-15 %	5-15 %	5-15 %	5-15 %
Increased Network Latency	50-200 ms	50-200 ms	50-200 ms	50-200 ms
Decreased Network Bandwidth	120-190 Mbps	15-30 Mbps	30-45 Mpbs	12-20 Mbps
HDD Stress	—	—	—	1-3 workers
Network Stress	—	—	—	line speed

Table II
PARAMETER VALUE RANGES OF INJECTED ANOMALIES

Anomaly Type	Detection Rate		Detection Latency	
	Physical Hosts	Virtual Hosts	Physical Hosts	Virtual Hosts
CPU Stress	83.4%	100%	<0.01s	1.72s ±27.88s
Memory Leak	76.5%	100%	43.30s ±112.42s	22.86s ±65.61s
Increased Network Latency	100%	100%	14.80s ±33.32s	1.21s ±2.67s
Packet Loss	100%	100%	21.22s ±94.71s	0.33s ±0.78s
Decreased Network Bandwidth	100%	100%	<0.01s	<0.01s
Packet Duplication	100%	100%	55.20s ±64.22s	0.46s ±1.08s
HDD Stress	—	100%	—	0.57s ±1.50s
Network Stress	—	100%	—	0.19s ±0.26s

Table III
EVALUATION: ANOMALY DETECTION RATE AND LATENCY

REFERENCES

[1] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," in *ACM Sigmod Record*, vol. 25, no. 2. ACM, 1996, pp. 103–114.

[2] A. Gulenko, M. Wallschläger, F. Schmidt, O. Kao, and F. Liu, "A system architecture for real-time anomaly detection in large-scale nfv systems," *Procedia Computer Science*, vol. 94, pp. 491–496, 2016, the 11th International Conference on Future Networks and Communications (FNC 2016).

[3] G. Silvestre, C. Sauvanaud, M. Kaâniche, and K. Kanoun, "An anomaly detection approach for scale-out storage systems," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2014 IEEE 26th International Symposium on*. IEEE, 2014, pp. 294–301.

[4] G. Silvestre, C. Sauvanaud, M. Kaâniche, and K. Kanoun, "Tejo: A supervised anomaly detection scheme for newsql databases," in *Software Engineering for Resilient Systems - 7th International Workshop, SERENE 2015, Paris, France, September 7-8, 2015. Proceedings*, 2015, pp. 114–127.

[5] M. Miyazawa, M. Hayashi, and R. Stadler, "vnmf: Distributed fault detection using clustering approach for network function virtualization," in *IM*, R. Badonnel, J. Xiao, S. Ata, F. D. Turck, V. Groza, and C. R. P. dos Santos, Eds. IEEE, 2015, pp. 640–645.

[6] C. Sauvanaud, K. Lazri, M. Kaâniche, and K. Kanoun, "Anomaly detection and root cause localization in virtual network functions," in *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*. IEEE, 2016, pp. 196–206.

[7] D. Cotroneo, R. Natella, and S. Rosiello, "A fault correlation approach to detect performance anomalies in virtual network function chains," in *Software Reliability Engineering (ISSRE), 2017 IEEE 28th International Symposium on*. IEEE, 2017, pp. 90–100.

[8] A. W. Williams, S. M. Pertet, and P. Narasimhan, "Tiresias: Black-box failure prediction in distributed systems," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE, 2007, pp. 1–8.

[9] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan, "Prepare: Predictive performance anomaly prevention for virtualized cloud systems," in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. IEEE, 2012, pp. 285–294.

[10] O. Vallis, J. Hochenbaum, and A. Kejariwal, "A novel technique for long-term anomaly detection in the cloud." in *HotCloud*, 2014.

[11] H. Kang, H. Chen, and G. Jiang, "Peerwatch: a fault detection and diagnosis tool for virtualized consolidation systems," in *ICAC*, 2010.

[12] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, "Statistical techniques for online anomaly detection in data centers," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*. IEEE, 2011, pp. 385–392.

[13] P. Xiong, C. Pu, X. Zhu, and R. Griffith, "vperfguard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ACM, 2013, pp. 271–282.

[14] D. J. Dean, H. Nguyen, and X. Gu, "Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems," in *Proceedings of the 9th international conference on Autonomic computing*. ACM, 2012, pp. 191–200.

[15] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, no. 1-3, pp. 1–6, 1998.

[16] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehörster, and A. Brinkmann, "Non-intrusive virtualization management using libvirt," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 574–579.

[17] B. Pfaff and B. Davie, "The open vswitch database management protocol," 2013. [Online]. Available: https://tools.ietf.org/html/rfc7047

[18] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, 2015, pp. 117–130.

[19] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[20] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *Journal of Machine Learning Research*, vol. 11, no. May, pp. 1601–1604, 2010.

[21] S. H. Makhsous, A. Gulenko, O. Kao, and F. Liu, "High available deployment of cloud-based virtualized network functions," in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 2016, pp. 468–475.