IAD: Indirect Anomalous VMMs Detection in the Cloud-based Environment

Anshul Jindal[1][0000−0002−7773−5342], Ilya Shakhat[2], Jorge Cardoso[2,3][0000−0001−8992−3466], Michael Gerndt[1][0000−0002−3210−5048], and Vladimir Podolskiy[1][0000−0002−2775−3630]

[1] Chair of Computer Architecture and Parallel Systems,
Technical University of Munich, Garching, Germany
anshul.jindal@tum.de, gerndt@in.tum.de, v.podolskiy@tum.de
[2] Huawei Munich Research Center, Huawei Technologies Munich,Germany
{ilya.shakhat1, jorge.cardoso}@huawei.com
[3] University of Coimbra, CISUC, DEI, Coimbra, Portugal

Abstract. Server virtualization in the form of virtual machines (VMs) with the use of a hypervisor or a Virtual Machine Monitor (VMM) is an essential part of cloud computing technology to provide infrastructure-as-a-service (IaaS). A fault or an anomaly in the VMM can propagate to the VMs hosted on it and ultimately affect the availability and reliability of the applications running on those VMs. Therefore, identifying and eventually resolving it quickly is highly important. However, anomalous VMM detection is a challenge in the cloud environment since the user does not have access to the VMM.

This paper addresses this *challenge of anomalous VMM detection in the cloud-based environment without having any knowledge or data from VMM* by introducing a novel machine learning-based algorithm called **IAD**: **I**ndirect **A**nomalous VMMs **D**etection. This algorithm solely uses the VM's resources utilization data hosted on those VMMs for the anomalous VMMs detection. The developed algorithm's accuracy was tested on four datasets comprising the synthetic and real and compared against four other popular algorithms, which can also be used to the described problem. It was found that the proposed *IAD* algorithm has an average F1-score of 83.7% averaged across four datasets, and also outperforms other algorithms by an average F1-score of 11%.

Keywords: anomaly detection · cloud computing · VMM · hypervisor.

1 Introduction

Cloud computing enables industries to develop and deploy highly available and scalable applications to provide affordable and on-demand access to compute and storage resources. Server virtualization in the form of virtual machines (VMs) is an essential part of cloud computing technology to provide infrastructure-as-a-service (IaaS) with the use of a hypervisor or Virtual Machine Monitor (VMM) [12]. Users can then deploy their applications on these VMs with only

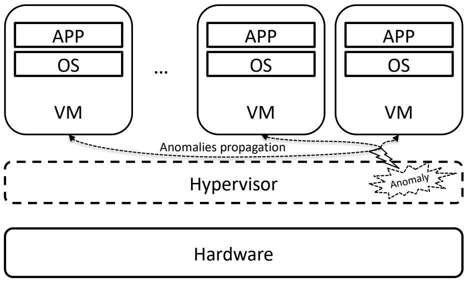


Fig. 1: An example showcasing the propagation of anomalies in a Type-1 hypervisor or VMM to the virtual machines (VMs) hosted on it.

the required resources. This allows the efficient usage of the physical hardware and reduces the overall cost. The virtualization layer, especially the hypervisors, is prone to temporary hardware errors caused by manufacturing defects, a sudden increase in CPU utilization caused by some task or disconnection of externally mounted storage devices, etc. The VMs running on these VMMs are then susceptible to errors from the underneath stack, as a result, can impact the performance of the applications running on these VMs [7,8]. Figure 1 shows an example propagation of anomalies in a virtualization stack using a type-1 hypervisor to the VM hosted on it. These anomalies may lead to the failure of all VMs and, ultimately, the applications hosted on them.

In the development environment, these anomalous VMMs are relatively easily detectable by analyzing the logs from the hypervisor dumps. But in the production environment running on the cloud, anomalous VMMs detection is a challenge since a cloud user does not have access to the VMMs logs. Additionally, many anomalous VMM detection techniques have been proposed [11,13,15]. However, these works either require the monitoring data of the hypervisor or inject custom probes into the hypervisor. Therefore, the usage of such solutions becomes infeasible. Furthermore, due to the low downtime requirements for the applications running on the cloud, detecting such anomalous VMMs and their resolutions is to be done as quickly as possible.

Therefore, this challenge is addressed in this paper for detecting anomalous VMMs *by solely using the VM's resources utilization data hosted on those VMMs* by creating a novel algorithm called **IAD**: **I**ndirect **A**nomalous VMMs **D**etection. We call the algorithm indirect since the detection must be done without any internal knowledge or data from the VMM; it should be solely based on the virtual machine's data hosted on it. The key contributions are :

- We present an online novel machine learning-based algorithm **IAD** for accurate and efficient detection of anomalous VMMs by solely using the resource's utilization data of the VM's hosted on them as the main metric (§3).
- We evaluate the performance of the *IAD* on two different aspects: 1) Anomalous VMMs finding accuracy (§5.1), and 2) Anomalous VMMs finding effi-

Table 1: Symbols and definitions.

Symbol	**Interpretation**
n	Number of time ticks in data
d	Number of virtual machines hosted on a VMM
X_t	The percentage utilization of a resource (for example, CPU or disk usage) by a VM at a time t
X_t^j	The percentage utilization of a resource at a time t for j^{th} VM
$\{c_t^1, c_t^2, ..., c_t^m\}$	a set of m $\leq$d VMs with change point at time tick t
w	Window size
minPercentVMsFault	Minimum % of total number of VMs on a VMM which must have a change point for classifying the VMM anomalous.

ciency and scalability (§5.2), and compare it against five other popular algorithms which can also be applied to some extent on the described problem.

- We evaluate the *IAD* algorithm and other five popular algorithms on synthetic and two real datasets.

Paper Organization: Section 2 describes the overall problem statement addressed in this paper along with an illustrative example. The design and details of the proposed *IAD* algorithm are presented in Section 3 . Section 4 provides experimental configuration details along with the algorithms and the datasets used in this work for evaluation. In Section 5, the evaluation results are presented. Finally, Section 6 concludes the paper and presents an outlook.

2 Problem Definition

This section presents the overall problem definition of indirectly detecting anomalous VMMs in a cloud-based environment. Table 1 shows the symbols used in this paper.

We are given $X = n \times d$ dataset, with n representing the number of time ticks and d the number of virtual machines hosted on a VMM. X_t^j denotes the percentage utilization of a resource (for example, CPU or disk usage) at a time t for j^{th} VM. Our goal is to detect whether the VMM on which the d virtual machines are hosted is anomalous or not. Formally:

Problem 1. (Indirect Anomalous VMM Detection)

- **Given** *a multivariate dataset of n time ticks, with d virtual machines (X_t^j for $j = \{1, \cdots, d\}$ and $t = \{1, \cdots, n\}$) representing the CPU utilization observations of VMs hosted on a VMM.*
- **Output** *a subset of time ticks or a time tick where the behavior of the VMM is anomalous.*

One of the significant challenges in this problem is the online detection, in which we receive the data incrementally, one time tick for each VM at a time, i.e., $X_1^j, X_2^j, \cdots$, for the j^{th} VM. As we receive the data, the algorithm should

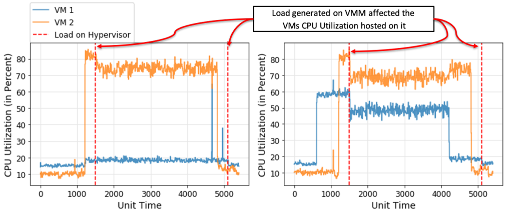


Fig. 2: Examples showing CPU utilization of two virtual machines hosted on a VMM. The left sub-figure shows an application running only on VM 2, while the right sub-figure shows the application running on both VMs. We can see a significant decrement in the CPU utilization of the two VMs when an anomaly (high-CPU load) is generated on the VMM (shown by dotted red lines).

output the time ticks where the behavior of the VMM is observed as anomalous. However, without looking at the future few time ticks after time t, it would be impractical to determine whether at time point t, the VMM is anomalous or not since the time ticks $t+1, t+2, \cdots$, are essential in deciding whether an apparent detection at time t was an actual or simply noise. Hence, we introduce a window parameter w, upon receiving a time tick $t+w$, the algorithm outputs whether at time t the VMM showcased anomalous behavior or not. Additionally, as the change points for VMs hosted on VMM could be spread over a specific duration due to the effect of the actual fault being propagating to the VMs and the granularity of the collected monitoring data, therefore, using an appropriate window size can provide a way for getting those change points.

2.1 Illustrative Example

Here we illustrate the problem with two examples in Fig. 2 showcasing the CPU utilization of two virtual machines hosted on a VMM. In the left sub-figure, an application is running only on VM 2, while in the right, an application is running on both VMs. During the application run time, an anomaly, i.e., high CPU load, was generated on the hypervisor for some time (shown by dotted red lines). During this time, we can observe a significant drop in the CPU utilization by the application (affecting the performance of the application) of the two VMs (especially when an application is running on the VM). The load on a VMM affects all or most of the VMs hosted on it, which ultimately can significantly affect the performance of the applications running on the two VMs; therefore, we call such a VMM anomalous when the load was generated on it.

3 Indirect Anomaly Detection (IAD) Algorithm

This section presents our proposed Indirect Anomaly Detection (IAD) algorithm along with the implemented system for evaluating it. The overall system workflow diagram is shown in Figure 4 and mainly consists of two parts: the main *IAD Algorithm*, and the *Test Module* for evaluating the algorithm.

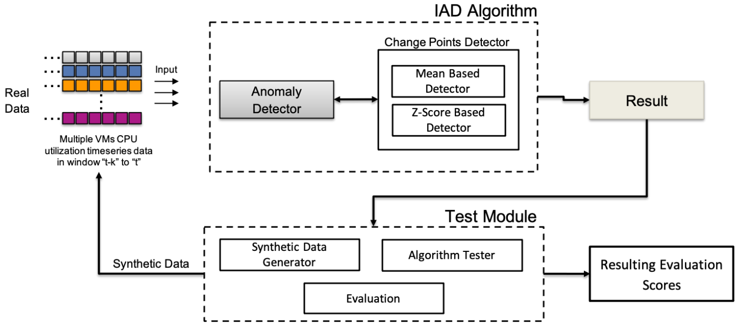


Fig. 3: High-level system workflow of the implemented system for evaluating IAD algorithm and the interaction between its components in a general use case.

3.1 IAD Algorithm

Our principal intuition behind the algorithm is that if a time tick t represents a change point for some resource utilization (such as CPU utilization) in most VMs hosted on a VMM; then the VMM is also anomalous at that time tick. This is based on the fact that a fault in VMM will affect most of the VMs hosted on it, and therefore those VMs would observe a change point at a similar point of time (in the chosen window w (Table 1)) in their resource's utilization. IAD algorithm consists of two main parts, described below:

Change Points Detector : We first explain how the change point, i.e., time tick where the time series changes significantly, is calculated. Recall from §2 that, we have introduced a window parameter w, upon receiving the time tick $t + w$, the *Change Points Detector* outputs whether the time tick t is a change point or not. Given a dataset X^j of size w for j^{th} VM, this component is responsible for finding the change points in that VM. This can be calculated in two ways: Mean-based detector and Z-score-based detector.

- **Mean-based Detector**: In this detector, a *windowed_mean*, i.e., the mean of all the values in the window, and the *global_mean*, i.e., the mean of all the values until the current time tick is calculated. Since the IAD algorithm is designed for running it in an online way, therefore not all the values can be stored. Thus *global_mean* is calculated using Knuth's algorithm [5,9]. We then calculate the absolute percentage difference between the two means: *windowed_mean* and *global_mean*. If the percentage difference is more significant than the specified threshold (by default is 5%), then the time tick t for j^{th} VM is regarded as the change point.
- **Z-score-based Detector**: This detector is based on the calculation of the Z-scores [4,6]. Similar to the Mean-based detector, here also a *windowed_mean*, i.e., the mean of all the values in the window, and the *global_mean*, i.e., the mean of all the values until the current time tick is calculated. We additionally calculate the *global_stand_deviation*, i.e., the standard deviation of all

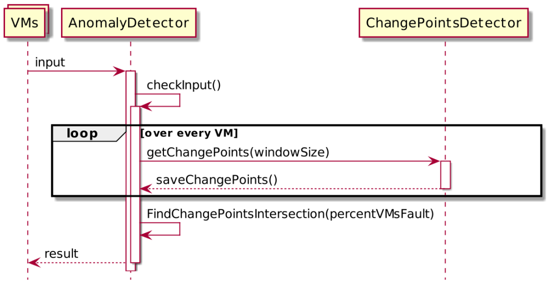


Fig. 4: Indirect Anomaly Detection (IAD) Algorithm workflow sequence diagram

the values until the current time tick. Since the IAD algorithm is designed for running it in an online way, *global_stand_deviation* is calculated using Welford's method [9]. These statistics are then used for the calculation of the z-scores for all the data points in the window using Equation 1.

$$z\_scores = \frac{(windowed\_mean - global\_mean)}{\frac{global\_stand\_deviation}{\sqrt{w}}} \qquad (1)$$

If the Z-scores of all windowed observations are greater than the defined threshold (3 × *global_stand_deviation*) then the time tick t for j^{th} VM is regarded as the change point.

In the main algorithm, only *Z-Ssore-based Detector* is used as it provides higher accuracy and has fewer false positives.

Anomaly Detector This component receives the input resource utilization data X of size $n \times d$ where d is the number VMs hosted on a VMM along with the `minPercentVMsFault` (Table 1)) as the input parameter. We first check the input timeseries of w length for 1) zero-length timeseries and 2) if the input timeseries of all VMs are of the same length or not. If any of the two initial checks are true, then we quit and don't proceed ahead. We assume that all the VM's resources utilization data is of the same length only. After doing the initial checks, each of the VM's windowed timeseries belonging to the VMM is sent to the *Change Points Detector* for the detection of whether the time tick t is a change point or not. If the percentage number of VMs ($\{c_t^1, c_t^2, ..., c_t^m\}$ out of d) having the change point at time tick t is greater than the `minPercentVMsFault` input parameter, then the VMM is reported as anomalous at time tick t. The above procedure is repeated for all time ticks. Figure 4 shows the workflow sequence diagram of the IAD algorithm. Furthermore, the developed approach can be applied for multiple VMMs as well.

3.2 Test Module

This component is responsible for generating the synthetic data and evaluating the algorithm performance by calculating the F1-score on the results from the algorithm. It consists of multiple sub-component described below:

- **Synthetic Data Generator**: It takes the number of VMMs, number of VMs per VMM, percentage of the VMs with a fault; as the input for generating synthetic timeseries data. This synthetic data follows a Gaussian distribution based on the input parameters. This component also automatically divides the generated data into true positive and true negative labels based on the percentage of the VMs with a fault parameter.
- **Algorithm Tester**: It is responsible for invoking the algorithm with various parameters on the synthetic data and tune the algorithm's hyperparameters.
- **Evaluation**: The results from the algorithm are passed as the input to this sub-component, where the results are compared with the actual labels, and the overall algorithm score in terms of F1-score is reported.

4 Experimental Settings

We design our experiments to answer the following questions:

Q1. Indirect Anomaly Detection Accuracy: how accurate is IAD in the detection of anomalous VMM when compared to other popular algorithms?

Q2. Anomalous VMMs finding efficiency and scalability: How does the algorithm scale with the increase in the data points and number of VMs?

4.1 Datasets

For evaluating the IAD algorithm, we considered four types of datasets listed in Table 2 along with their information and are described below:

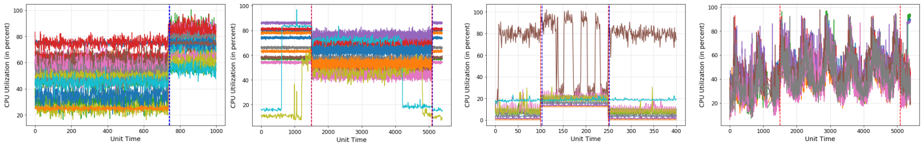
Synthetic: This is the artificially generated dataset using the *Test Module* component described in §3.

Experimental-Synthetic Merged: This is a dataset with a combination of experimental data and synthetic data. We created two nested virtual machines on a VM in the Google Cloud Platform to collect the experimental dataset. The underneath VM instance type is n1-standard-4 with four vCPUs and 15 GB of memory, and Ubuntu 18.04 OS was installed on it. This VM instance acts as a host for the above VMs. *libvirt* toolkit is used to manage and create nested virtualization on top of the host machine. Kernel-based Virtual Machine (KVM) is used as a VMM. The configuration of the two nested VMs are i) 2vCPU and 2GB memory, ii) 1vCPU and 1GB memory. Cloud-native web applications were run on these two VMs. Monitoring data from the two VMs and underneath host is exported using the Prometheus agent deployed on each of them to an external virtual machine. *stress-ng* is used for generating the load on the VMM. Based on this infrastructure, we collected a dataset for various scenarios and combined it with the synthetic data.

Table 2: Datasets used in this work for evaluating the algorithms.

Dataset Name	**Anomalous VMMs**	**Non-Anomalous VMMs**	**VMs Per VMM**	**TimeTicks per VM**
Synthetic	5	5	10	1000
Exp-Synthetic Merged	42	17	2 (experimental) 8 (synthetic)	5400
Azure† [1]	16	10	10	5400
Alibaba† [14]	10	10	10	5400

†These are modified for our usecase.



(a) Synthetic (b) Exp-Synthetic (c) Azure (d) Alibaba

Fig. 5: An example profile of an anomalous VMM having 10 VMs in all the datasets used in this work for evaluation.

Azure Dataset: This dataset is based on the publicly available cloud traces data from Azure [1]. We used the VMs data from it and created random groups of VMs, with each group representing the VMs hosted on a VMM. Afterward, we feed these timeseries groups in our synthetic data generator for randomly increasing or decreasing the CPU utilization of the VMs within a VMM based on the input parameters to create anomalous and non-anomalous VMMs.

Alibaba Dataset: This dataset is based on the publicly available cloud traces and metrics data from Alibaba cloud [14]. A similar method as the *Azure Dataset* was also applied to form this dataset.

Figure 5 shows an example profile of an anomalous VMM for all the datasets.

4.2 Evaluated Algorithms

We compare IAD to the five other algorithms listed in Table 3 along with their input dimension and parameters. ECP is a non-parametric-based change detection algorithm that uses the E-statistic, a non-parametric goodness-of-fit statistic, with hierarchical division and dynamic programming for finding them [3]. BnB (Branch and Border) and its online version (BnBO) are also non-parametric change detection methods that can detect multiple changes in multivariate data by separating points before and after the change using an ensemble of random partitions [2]. Lastly, we use the popular anomaly detection algorithm: isolation forest for detecting anomalous VMM [10]. The primary isolation forest (IF) works on the input data directly, while we also created a modified version of it called the isolation forest features (IFF), which first calculates several features

Table 3: The details of the algorithms used in this work for evaluation, along with their input dimension and parameters.

Algorithm	Input Dimension	Parameters
IAD	n × d	w, minPercentVMsFault
ECP [3]	n × d	change points, Min. points b/w change points
BNB [2]	n × d	w, number of trees, threshold for change points
BNBOnline [2]	n × d	w, number of trees, threshold for change points
IF [10]	n × d	contamination factor (requires training)
IFF [10]	n × features	contamination factor (requires training)

such as mean, standard deviation, etc., for all values within a window on the input dataset and then apply isolation forest on it. The downside of the IF and IFF is that they require training.

4.3 Other Settings

We have used F1-Score (denoted as F1) to evaluate the performance of the algorithm. Evaluation tests have been executed on 2.6 GHz 6-Core Intel Core i7 MacBook Pro, 32 GB RAM running macOS BigSur version 11. We implement our method in Python. For our experiments, hyper-parameters are set as follows. The window size w is set as 1 minute (60 samples, with sampling done per second), threshold k as 5%, and percentVMsFault f as 90%. However, we also show experiments on parameter sensitivity in this section.

5 Results

Our Initial experiments showed that 1) CPU metric is the most affected and visualized parameters in the VMs when some load is generated on the VMM; 2) All or most VMs are affected when a load is introduced on the VMM.

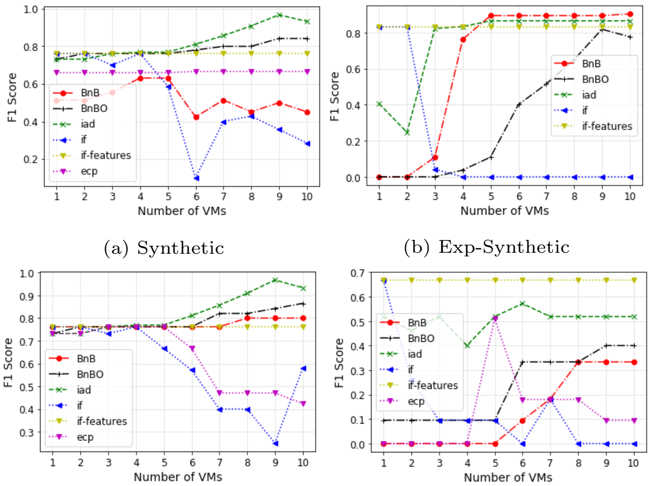
5.1 Q1. Indirect Anomaly Detection Accuracy

Table 4 shows the best F1-score corresponding to each algorithm evaluated in this work (§4.2) and on all the datasets (§4.1). We can observe that *IAD* algorithm outperforms the others on two datasets, except for the Experiment-Synthetic dataset (BNB performed best with F1-Score of `0.90`) and Alibaba dataset (IFF performed best with F1-Score of `0.66`. However, if one wants to find an algorithm that is performing well on all the datasets (Average F1-score column in Table 4), in that case, *IAD* algorithm outperforms all the others with an average F1-score of `0.837` across all datasets.

Furthermore, we present the detailed results of the algorithms on all four datasets varying with the number of VMs and are shown in Figure 6. One can observe that *IAD* performs best across all the datasets, and its accuracy increases with the increase in the number of VMs. Additionally, after a certain number of

Table 4: F1-score corresponding to each algorithm evaluated in this work (§4.2) and on all the datasets (§4.1)

Algorithm	Synthetic	Exp-Synthetic	Azure	Alibaba	Average F1-score
IAD	**0.96**	0.86	**0.96**	0.57	**0.837**
ECP	0.67	-	0.76	0.51	0.64
BNB	0.62	**0.90**	0.8	0.33	0.662
BNBOnline	0.87	0.81	0.86	0.4	0.735
IF	0.76	0.83	0.76	0.2	0.637
IF Features (IFF)	0.76	0.83	0.76	**0.66**	0.75



(a) Synthetic

(b) Exp-Synthetic

(c) Azure

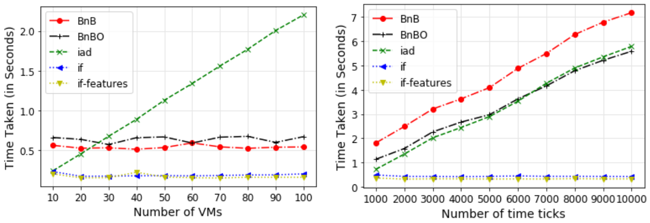
(d) Alibaba

Fig. 6: F1-score variation with the number of VMs corresponding to each algorithm evaluated in this work (§4.2) and on all the datasets (§4.1)

VMs, the F1-score of *IAD* becomes stable. This shows that if, for example, we have the synthetic dataset, then the best performance is possible with VMs ≥ 9. Similarly, in the case of the Azure dataset, while for the Exp-Synthetic dataset, one needs at least five VMs, and for the Alibaba dataset, seven VMs for the algorithm to perform well.

5.2 Q2. Anomalous VMMs finding efficiency and scalability

Next, we verify that our algorithm's detection method scale linearly and compare it against other algorithms. This experiment is performed with the synthetic dataset, since we can increase the number of VMs per VMM in it. We linearly increased the number of VMs from 1 to 100 and repeatedly duplicated our dataset in time ticks by adding Gaussian noise. Figure 7 shows various algorithm's detection method scalability for different parameters. One can observe that *IAD's*



(a) With number of VMs

(b) With number of time ticks

Fig. 7: Algorithm's detection method scalability with respect to different parameters.

detection method scale linearly in terms of both the parameters. However, when the number of VMs are scaled to 100, *IAD* takes a longer time as compared to others, but it provides results under 2.5s which if we see is not that much considering the accuracy we get with that algorithm. However, on the time ticks parameter, *BNB*, *BNBOnline* and *IAD* performed similar to each other, while *IF* and *IFF* provides results under 1 second, but its accuracy is worse as compared to the others on all the datasets, and it has the extra overhead of training. *ECP* algorithm's results are not shown, since it requires more than an hour for performing the detection with 100 VMs and 100,000 time ticks.

6 Conclusion

We propose *IAD* algorithm for indirect detection of anomalous VMMs by solely using the resource's utilization data of the VM's hosted on them as the primary metric. We compared it against the popular change detection algorithms, which could also be applied to the problem. We showcased that *IAD* algorithm outperforms all the others on an average across four datasets by 11% with an average accuracy score of 83.7%. We further showcased that *IAD* algorithm scale's linear with the number of VMs hosted on a VMM and number of time ticks. It takes less than 2.5 seconds for *IAD* algorithm to analyze 100 VMs hosted on a VMM for detecting if that VMM is anomalous or not. This allows it to be easily usable in the cloud environment where the fault-detection time requirement is low and can quickly help DevOps to know the problem is of the hypervisor or not.

The future direction includes using other metrics like network and storage utilization to enhance the algorithm's accuracy further.

References

1. Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., Bianchini, R.: Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In: Proceedings of the 26th Symposium on Operating Systems Principles. p. 153–167. SOSP '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3132747.3132772, https://doi.org/10.1145/3132747.3132772

2. Hooi, B., Faloutsos, C.: Branch and border: Partition-based change detection in multivariate time series. In: SDM (2019)
3. James, N.A., Matteson, D.S.: ecp: An r package for nonparametric multiple change point analysis of multivariate data (2013)
4. Jindal, A., Gerndt, M., Bauch, M., Haddouti, H.: Scalable infrastructure and workflow for anomaly detection in an automotive industry. In: 2020 International Conference on Innovative Trends in Information Technology (ICITIIT). pp. 1–6 (2020). https://doi.org/10.1109/ICITIIT49094.2020.9071555
5. Knuth, D.E.: The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms. Addison-Wesley Longman Publishing Co., Inc., USA (1997)
6. Kochendörffer, R.: Kreyszig, e.: Advanced engineering mathematics. j. wiley & sons, inc., new york, london 1962. ix + 856 s. 402 abb. preis s. 79.—. Biometrische Zeitschrift **7**(2), 129–130 (1965). https://doi.org/https://doi.org/10.1002/bimj.19650070232, `https://onlinelibrary.wiley.com/doi/abs/10.1002/bimj.19650070232`
7. Le, M., Tamir, Y.: Rehype: Enabling vm survival across hypervisor failures. In: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. p. 63–74. VEE '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/1952682.1952692, `https://doi.org/10.1145/1952682.1952692`
8. Li, M.L., Ramachandran, P., Sahoo, S.K., Adve, S.V., Adve, V.S., Zhou, Y.: Understanding the propagation of hard errors to software and implications for resilient system design. In: ASPLOS 2008 (2008)
9. Ling, R.F.: Comparison of several algorithms for computing sample means and variances. Journal of the American Statistical Association **69**(348), 859–866 (1974). https://doi.org/10.1080/01621459.1974.10480219, `https://www.tandfonline.com/doi/abs/10.1080/01621459.1974.10480219`
10. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation forest. In: 2008 Eighth IEEE International Conference on Data Mining. pp. 413–422 (2008). https://doi.org/10.1109/ICDM.2008.17
11. Nikolai, J., Wang, Y.: Hypervisor-based cloud intrusion detection system. 2014 International Conference on Computing, Networking and Communications (ICNC) pp. 989–993 (2014)
12. Parashar, M., AbdelBaky, M., Rodero, I., Devarakonda, A.: Cloud paradigms and practices for computational and data-enabled science and engineering. Computing in Science Engineering **15**(4), 10–18 (2013). https://doi.org/10.1109/MCSE.2013.49
13. Reinhardt, S.K., Mukherjee, S.S.: Transient fault detection via simultaneous multithreading. In: Proceedings of the 27th Annual International Symposium on Computer Architecture. p. 25–36. ISCA '00, Association for Computing Machinery, New York, NY, USA (2000). https://doi.org/10.1145/339647.339652, `https://doi.org/10.1145/339647.339652`
14. Shan, Y., Huang, Y., Chen, Y., Zhang, Y.: Legoos: A disseminated, distributed os for hardware resource disaggregation. In: Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation. p. 69–87. OSDI'18, USENIX Association, USA (2018)
15. Xu, X., Chiang, R.C., Huang, H.H.: Xentry: Hypervisor-level soft error detection. In: 2014 43rd International Conference on Parallel Processing. pp. 341–350 (2014). https://doi.org/10.1109/ICPP.2014.43