Detection of Service Provider Hardware Over-commitment in Container Orchestration Environments

Pedro Horchulhack*, Eduardo K. Viegas*†, Altair O. Santin*
*Pontifícia Universidade Católica do Paraná (PUCPR) — Graduate Program in Computer Science (PPGIa)
Curitiba, Paraná, 80215-901, Brazil.
{*pedro.horchulhack, santin*}*@ppgia.pucpr.br*
†Secure Systems Research Center, Technology Innovation Institute, Abu Dhabi 9639, United Arab Emirates.
eduardo@ssrc.tii.ae

GLOBECOM 2022 - 2022 IEEE Global Communications Conference | 978-1-6654-3540-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/GLOBECOM48099.2022.10001375

***Abstract*—The deployment of container-based services continues to increase as time passes, mainly due to its fast provision time and lower allocation overheads. Yet, the literature still neglects the performance degradation in containers due to multi-tenancy and service provider hardware over-commitment. This paper proposes a new hardware over-commitment detection for container orchestration environments, implemented twofold. First, the containerized hardware usage of deployed containers is continuously monitored in a non-intrusive manner, leveraging the container engine resource management interface. Second, collected features are used by a recurrent neural network model for detecting both container and service level hardware over-commitment, following a time-series rationale. Experiments run on a containerized Apache Spark distribution have shown that multi-tenancy and hardware over-commitment significantly affect its performance. In addition, our proposed model is able to detect hardware over-commitment with up to 91% of true-positive at the container level, and up to 93% true-positive at the service level.**

***Index Terms*—Container, Multi-tenancy, Recurrent Neural Network, Kubernetes, Docker.**

I. Introduction

The cloud computing service model has relied on hardware sharing through a hypervisor [1]. The software enables the provision of a virtual machine (VM) executed on top of a virtualized layer of the provider's physical hardware resources [2]. The tenant, in turn, executes on the VM the desired operating system (OS) and his needed applications. As a result, the VM provision and management task introduces significant allocation overheads, as a new OS must be instantiated for each provisioned VM [3].

To adequately address the cloud elasticity requirements, computational services initially deployed through VMs have been increasingly shifted to container-based solutions [3]. The container is executed within the host OS environment as an isolated process. Thus, it can leverage the host OS libraries and resources, demanding significantly less computational resources and provision time [4].

The cloud computing service model relies on multi-tenancy for the sharing of the provider physical hardware resources between several clients. The virtualized hardware resources are provided through a *pay-as-you-go* model, wherein clients only pay for the hardware resource they use, possibly leaving underutilized resources to be allocated to other cloud tenants [5]. In such a context, cloud providers often overcommit their physical hardware resources to other cloud tenants for profit purposes, assuming that cloud tenants will not simultaneously demand their allocated virtualized hardware [6].

Multi-tenancy introduces significant performance degradation on VMs. Thus, hypervisors have been continuously improved over the past years to adequately address the fair sharing of resources between allocated tenants. For example, the CPU scheduler used by the VMWare hypervisor employs a CPU share scheme which is continuously evaluated during the CPU assignment task, giving higher priority to tenants with higher CPU shares [7]. As a result, even if a shortage of physical hardware resources occurs due to resource overcommitment, the hypervisor can ensure fair hardware access, decreasing the impact on the tenant quality-of-service (QoS). In contrast, container-based applications often rely on the host's kernel scheduler, which may defer the container access to the physical hardware to pave the way for a higher privileged host OS process [1]. The literature overlooks such a challenge, assuming that the impact of multi-tenancy in containers is similar to those experienced in traditional VM-based clouds

Cloud computing providers continuously monitor the performance of allocated client services to ensure that the service level agreements (SLA) are adequately met [8], [9]. Over the last years, several works have been proposed for VM performance monitoring in a non-intrusive way, hence, without requiring access to the client's isolated user space. Surprisingly, container performance monitoring is still mostly overlooked in the literature [10]. It is assumed that container-based services will be subject to the same multi-tenancy interference experienced on VM-based deployments, neglecting the resource management differences between the hypervisor and container.

This paper proposes a new model for performance degra-

dation detection in multi-tenant container-based environments, implemented twofold. First, we continuously and periodically monitor the hardware usage of distributed deployed containers in a non-intrusive manner. Our proposed model does not require access to the isolated container environment and can be executed on top of the provider host OS. Second, the collected hardware usage metrics of the monitored containers are used by a recurrent neural network (RNN) model to detect container and service level performance degradation. Our proposed model can detect performance degradation due to multi-tenancy considering both container and service levels without demanding access to the isolated client space in a non-intrusive manner.

The main contributions of this paper are:

- An evaluation of the performance degradation due to hardware overcommitment in a distributed container-based deployment. Experiments using a distributed containerized Apache Spark as a use-case have shown that hardware overcommitment significantly affects deployed services' processing performance.
- A new model based on RNN for detection of performance degradation in multi-tenant containerized services. The proposed model can detect hardware overcommitment with up to 91% of true-positive at the container-level and up to 93% true-positive at the service level.

II. Preliminaries

A. Virtual Machine

Over the last decades, the cloud computing service model have mostly relied on virtual machine (VM) deployment to share the provider's physical hardware between several tenants [5]. Cloud provider uses the hypervisor software to create a virtualized abstraction of the underlying physical hardware. The VM access the physical hardware through the provided virtualized hardware layer, which is managed by the hypervisor [2]. The hypervisor manages the virtualized hardware access to the physical hardware, managing and ensuring the fair sharing of resources among several tenants.

Hypervisor scheduling algorithms have been extensively studied and improved to address the fair sharing of resources between cloud tenants [7]. Cloud providers have leveraged such improvements to increase their profits, even over-allocating their physical hardware to cloud tenants, thus, relying on the hypervisor to adequately address the hardware sharing, as well as decreasing performance degradation impact due to multi-tenancy.

Nowadays, VM-based deployment of services is unable to meet the modern cloud elasticity requirements. A VM requires the execution of a new operating system (OS) and the client services, which introduces significant virtualization overheads, increasing the provision time and wasting computational resources. In practice, the VM provision task may demand several minutes to be fulfilled by the cloud provider.

B. Container

In light of this, container-based solutions are being increasingly used to deploy computational services. In such a case, the needed application service is executed in a containerized manner, sharing the host OS libraries with other tenants. The container is isolated from the host OS and other containers while sharing the host OS libraries [11].

The host OS becomes responsible for conducting the resource allocation and provision tasks, which are generally executed and managed as an isolated host OS process managed by the host OS kernel. For example, docker-based containers in Linux-based OS are managed through namespaces and control groups (cgroups) [12].

C. Multi-tenancy and Hardware Overcommitment

Cloud providers rely on hardware multi-tenancy to provide their services, leaving cloud tenants to dispute over time to use the provider's physical hardware resources. Multi-tenancy may significantly affect the processing performance of deployed services as the cloud provider may over-allocate its physical hardware to use underutilized resources to other tenants for profit purposes [6]. In such a case, if deployed tenants concurrently demand hardware access, the provider will not be able to meet the hardware requirements, degrading the client QoS.

Multi-tenancy introduces computational costs associated with hardware management and sharing with several tenants. Thus, the hypervisor must manage the hardware access between deployed VMs reasonably. Performance degradation due to VM multi-tenancy has been a widely explored topic in the literature. Hypervisors have been continuously improved to overcome the performance impact introduced by sharing physical hardware resources among allocated VMs.

Despite the increase in container-based deployment of services, performance degradation due to multi-tenancy in containers remains overlooked in the literature [10]. Containers are subject to a higher performance degradation since they are typically executed as a host OS process, thus, managed by the host OS kernel, which does not consider fair resource sharing in multi-tenancy settings.

III. Related Works

Cloud providers generally monitor deployed tenants' QoS to ensure that the previously established service-level agreements (SLA) have been met [13]. Service-level indicators (SLI) must be periodically collected and evaluated while ensuring that the client isolation is kept [14]. In other words, the cloud provider must monitor the performance of allocated tenants without having access to or modifying the isolated client space, which includes the running OS, and deployed services.

Ntambu and Adeshina [15] tackled virtual machine resource usage in a cloud environment through machine learning techniques. Despite their proposed model reaching high classification results, the authors considered collecting data within the client domain. Huang *et al* [16] used a LSTM to identify performance degradation in VM environments

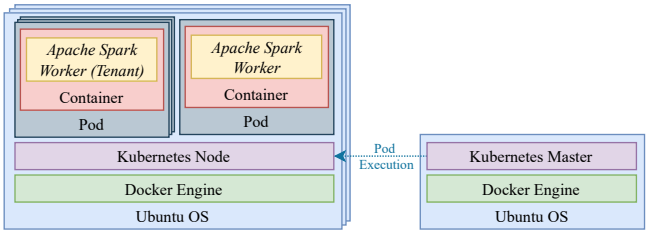


Fig. 1: Testbed architecture considering a distributed containerized Apache Spark job deployed through Kubernetes. Apache Spark jobs are deployed as Kubernetes Pods. Containerized multi-tenancy interference is created deploying additional Pods.

with relevant results. Similarly, the authors tackled the issue within the client domain. Podolskiy *et al.* [14] proposed a model to identify SLI deviations through machine learning techniques and evaluated their approach using data collected from a private Kubernetes cluster. Wang *et al.* [17] proposed a dynamic resource prediction over container metrics. However, the authors do not address the cloud provider's perspective. Also, regarding RNNs, Masouros *et al.* [18] used an LSTM to detect performance degradation of containerized applications.

IV. Problem Statement

In this section, we further investigate the impact of multi-tenancy in the container-based deployment of services. More specifically, we first describe the used testbed, considering a distributed containerized service, then how multi-tenancy affects the processing performance of the selected service.

A. Testbed

The implemented testbed is shown in Figure 1, it considers a container orchestration provider implemented through Kubernetes v.1.23.5. The provider is composed of 5 physical machines, wherein 4 machines act as Kubernetes nodes for deploying to-be-executed containers, and 1 machine acts as Kubernetes master. The Kubernetes nodes execute the containers through Docker v.20.10.7. The provider's physical machines are equipped with an 8-core Intel i7 CPU, 16GB of memory, interconnected through a gigabit network, running on top of an Ubuntu v.20.04 OS.

To evaluate the multi-tenancy performance impact, we consider a distributed containerized Big Data processing service [19]. More specifically, we consider a client that executes a containerized distributed Apache Spark job while being subject to performance interference caused by other tenants also running their own jobs (see Figure 1). In such a case, a TeraSort Apache Spark job was evaluated as implemented through the HiBench Suite [20]. The job randomly generates a series of values and orders them accordingly.

The Apache Spark architecture is executed as a Kubernetes Pod, composed of 4 Spark workers, each with 2 CPU cores and 1 Spark master. Each Kubernetes Pod (containerized Apache Spark job) is executed while varying the number of concurrent Pods (also running their own Apache Spark job). The multi-tenancy impact on our testbed is created by the concurrent execution of other Kubernetes Pods.

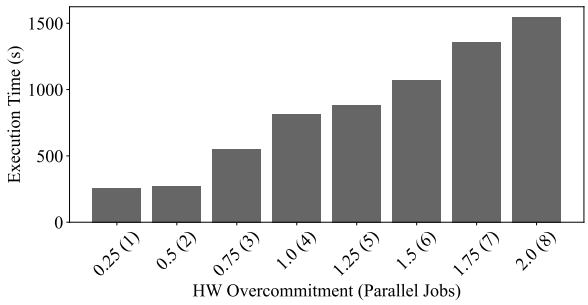


Fig. 2: Job processing time according to the number of concurrently deployed jobs in a multi-tenancy setting.

B. How does multi-tenancy affect the processing performance of containerized services?

Our evaluation aims to verify the job processing time while varying the number of concurrent executed Kubernetes Pod. More specifically, we vary the level of multi-tenancy interference over our deployed containerized service and evaluate the job processing time impact. Figure 2 shows the job processing time according to the number of parallel jobs.

Recalling that, due to the testbed configuration and the number of allocated CPU cores to each container, our hardware should be able to run 4 parallel jobs without impact on processing time, given the availability of HW resources. However, it is possible to note that multi-tenancy has a significant processing impact over the selected Apache Spark jobs, even in a 3 parallel job setting. In such a case, the job processing time increases by 175%. As a result, multi-tenancy significantly degrades the processing performance of containerized services.

The experiment showed that multi-tenancy significantly affects the performance of containerized services. This is because the container engine (Fig. 1, *Docker*), is not able to fairly address the resource sharing between multiple tenants, significantly degrading the processing performance even if the service provider is not over-committing its hardware. The processing performance of containerized services degrades in a higher multi-tenancy setting. The service provider must detect such processing degradation to take countermeasures, e.g., migrate a set of deployed containers to a different Kubernetes node.

V. A Recurrent Neural Network Model for Performance Degradation Detection due to Multi-tenancy

We present a detection model based on a recurrent neural network (RNN) to address the challenge of performance degradation due to multi-tenancy in containerized services. The proposal considers a container orchestration environment (e.g., Kubernetes) that monitors the QoS of deployed containers.

The monitoring goal is the identification of performance degradation due to multi-tenancy to ensure that SLAs are appropriately met. The provider monitoring task must be performed without having access to the isolated container user

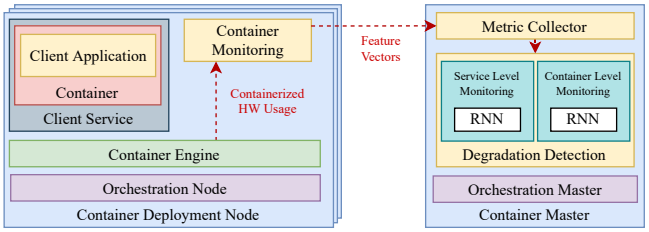


Fig. 3: The recurrent neural network model for detection of performance degradation in multi-tenant containerized environments.

space. Thus, the client can ensure that the provider does not have access to his data and running services. The provider must monitor the container performance without having access to the application data or the containerized OS usage metrics. The provider monitors the performance of deployed containers to identify performance degradation at both container and service levels. In other words, performance degradation due to multi-tenancy is identified for each deployed container and client distributed service.

The proposed model is shown in Figure 3, and is implemented through two main modules, namely *Container Monitor* and *Degradation Detection*. The *Container Monitor* is executed within each provider node used for the execution of client container. The module is executed periodically and collects the deployed container hardware usage metrics according to each hardware resource managed by the container engine, e.g., CPU, network, and disk.

Such an approach assumes that performance degradation can be identified according to deviations between the container hardware resources usage. For instance, a high CPU usage by a container is typically followed or preceded by an increase in network or disk usage. In contrast, in a performance degraded scenario, a high CPU usage might not increase network or disk usage, considering that the container will not be able to produce processing results accordingly. The metrics are collected at the node host level, thus, without accessing the isolated container space.

The collected metrics are sent to the *Degradation Detection* module, which goal is to detect container and service level performance degradation. The module compounds two sets of feature vectors according to the performance degradation level that will be identified. The built feature vectors are used as a representation of the current container orchestration environment, thus, used by a RNN model for performance degradation detection. The main insight of such an approach is that performance degradation is identified based on a time-series rationale, thus, taking into account the historical behavior of deployed containers.

The following subsections further describe our proposal, including the modules that implement it.

A. Container Monitor

The *Container Monitor* module goal is to periodically collect a set of container-related hardware usage metrics for further analysis. The module is executed within each provider node used to host client containers. The module collects hardware-related usage metrics for each container, e.g., CPU, network, and disk. The main assumption of our proposed model is that performance degradation can be identified according to deviations in each container's hardware resource usage.

The operation of the *Container Monitor* module is shown in Figure 3. The module is deployed at the provider nodes and periodically collects hardware usage metrics from each running container in a non-intrusive manner, thus, without affecting the container isolation, the module requests the container engine the container hardware usage in a given time interval, e.g., every 5 second. The collection of hardware usage metrics is performed within the host OS, as containers are often managed as a traditional host OS process so the module may monitor the container process metrics over time.

The collected set of hardware usage metrics is periodically sent for the *Degradation Module*.

B. Degradation Detection

The service client must ensure that the service provider does not have access to her data and running processes within the containerized domain. Therefore, the service provider monitoring task must be performed without violating the client isolation while evaluating the data collected at the container engine interface, which manages the container hardware access.

Container performance degradation affects the container's QoS of the running client services. On the other hand, service performance degradation may affect the QoS of the deployed client service according to the processing performance of the distributed client containers. Therefore, the *Degradation Detection* module goal is to detect performance degradation at both container and service levels.

The operation of the *Degradation Detection* module is shown in Figure 3. The module receives as input the extracted hardware usage metrics collected by the *Container Monitor* module (see Section V-A). The collected metrics are used to build two additional feature sets according to the performance degradation level that will be identified, concerning *container*, and *service* level, as follows:

- *Container*. Collected hardware usage metrics for each container are used as input by a container-level RNN model.
- *Service*. Collected hardware usage metrics for the set of client deployed containers are fed to a service-level RNN model.

Each used RNN model, including the *container-level* and *service-level*, performs the classification task through a time-series rationale. Our proposed scheme can identify performance degradation by taking into account the historical behavior of the deployed service, thus, increasing the system detection accuracy.

The proposed model can detect service performance degradation due to multi-tenancy without violating the client container isolation.

TABLE I: Features set extracted for each deployed container in a time interval of 5 seconds.

HW	***Feature***
CPU	User cycles, System cycles, Period number, Throttled Number, Throttled time
Memory	Resident Set Size, Cached, Mapped, Paged In, Paged Out, Page Faults, Major Page Faults, Active, Inactive, Active File, Inactive File, Unevictable
Disk	Read, Write, Sync, Async, Discarded, Total
Network	UP bytes, DL bytes, UP pkts., DL pkts., Transmission errors, Frame errors, Discarded pkts., Compressed pkts., Transmission losses, Multicast frames

VI. Evaluation

The evaluations aim at answering the following research questions (RQ): (***RQ1***) *How does our proposed model perform for detection of container-level performance degradation?* (***RQ2***) *Is our proposed model able to detect service performance degradation?* (***RQ3***) *How does the hardware overcommitment ratio affect the classification accuracy of our proposed scheme?*

The following subsections further describe our developed model prototype and its evaluation.

A. Prototype

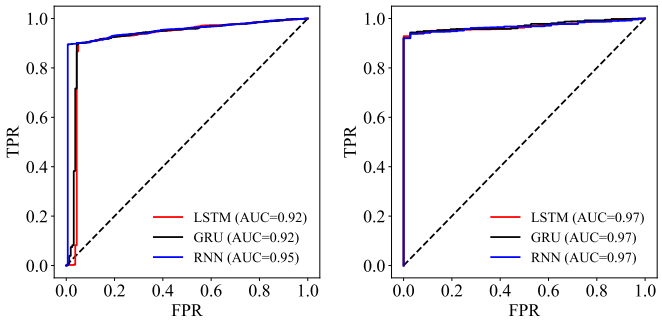
A proposal prototype was implemented and evaluated on top of our previously described Kubernetes cluster (see Section IV). The *Container Monitor* module is executed on top of each deployed Kubernetes node.

The module collects 5 CPU, 12 memory, 6 disk, and 10 network metrics, compounding a total of 33 hardware usage features for each deployed container, as shown in Table I. The features are collected every 5 second interval through the Linux *cgroups* (*/proc*).

The collected features are sent in real-time to the *Degradation Detection* module through a web service call implemented using SOAPpy API $v.0.12.22$. The built feature vectors were classified using three widely used recurrent neural network (RNN) architectures, namely Long-short Term Memory (LSTM), Gated Recurrent Unit (GRU), and a simple Recurrent Neural Network (RNN). The LSTM was implemented with 128 LSTM units, followed by a 2 unit dense layer. The GRU was implemented with 128 GRU units, followed by batch normalization and also a 2 unit dense layer. The RNN was implemented with 128 *simpleRNN* units and a 2 unit dense layer.

Each evaluated architecture was executed for $1,000$ epochs, and its learning rate set empirically according to the resulting loss, a momentum weight of 0.9, and using *adam* optimizer. The architectures parameters were defined similarly to related works [21]. The architectures were implemented through $keras$ API $v.2.4.0$, and $tensorflow$ API $v.2.4.1$.

The built dataset was split into *train*, *test*, and *validation*, each respectively composed by 40%, 30%, and 30% of the original dataset. The selected RNN models were evaluated concerning their true-positive (TP) and true-negative (TN) rates while considering positive samples as hardware overcommitment and normal samples as normal scenarios.



(a) Container-level (b) Job-level

Fig. 4: ROC curve of the proposed model for detecting HW overcommitment in multi-tenant container orchestration environments.

TABLE II: Proposal accuracies for HW overcommitment detection considering scenarios with less than 1.0 of HW overcommitment level as normal (see Figure 2).

Level	**RNN**	**Accuracies (%)**	
		TeraSort Job	
		TP	**TN**
Container	GRU	91.29	86.34
	LSTM	91.37	86.34
	RNN	91.86	83.85
Service	GRU	92.28	97.22
	LSTM	92.77	100.00
	RNN	93.11	97.22

The testbed label, *normal* or *HW overcommitment*, was established according to the hardware overcommitment level (see Figure 2). In such a case, samples collected in a scenario with less than 1.0 of hardware overcommitment were labeled as normal. In contrast, those generated in a scenario with more or equal 1.0 were labeled as HW overcommitment.

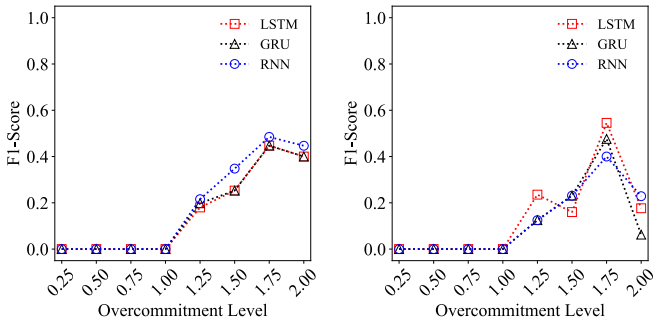
B. HW Overcommitment Detection

The first experiment aims at answering *RQ1*, and evaluates the accuracy of the proposed model for the detection of container-level HW overcommitment. Figure 4a shows the receiver operating characteristic (ROC) curve for detecting container-level performance degradation.

The proposed model provided up to 0.92, 0.92, and 0.95 AUC for detecting container-level performance degradation for the LSTM, GRU, and RNN architectures, respectively. In the best case, as shown in Table II, our proposed model can reach 91% of TP and 86% of TN. Therefore, our proposed scheme can detect container-level HW overcommitment without requiring access to the isolated container space.

The second experiment aims at answering *RQ2*, and evaluates the accuracy of our proposed scheme for the detection of performance degradation at the service level. We build a feature vector based on the concatenation of the feature sets extracted from all deployed containers for each job.

Figure 4b shows the obtained AUC, while Table II shows the classification accuracy of our proposed scheme for detecting service-level HW overcommitment. Our model provided high



(a) Container-level (b) Job-level

Fig. 5: The model accuracy according to the level of HW overcommitment in a multi-tenant container orchestration environment.

classification accuracies, with up to 0.97 of AUC, reaching a TP rate of 93% and a TN rate of 97%. Our proposal can detect service-level HW overcommitment in containerized services without requiring access to the isolated container user space.

Finally, to answer *RQ3*, we further investigate how the HW overcommitment level affects the classification accuracy of our model. Figure 5 shows the F1-Score for the proposal in both container-level and job-level HW overcommitment detection. Our proposed model can provide even higher classification accuracies when the HW overcommitment is low (≤ 1.0). This is because the extracted feature values become unstable when the HW overcommitment is high. Therefore, our proposed scheme can increase its classification accuracies when operated in a real-world environment, wherein the provider should provide the expected SLA.

VII. Conclusion

Performance degradation due to multi-tenancy in container-based deployment of services has been overlooked in the literature. This paper has shown that multi-tenancy significantly affects the processing performance of distributed containerized services. Notwithstanding, we have proposed and evaluated a new model based on recurrent neural network for the detection of HW overcommitment. Our proposed scheme can detect container and service level performance degradation without requiring access to the isolated container user space. In future works, we plan to extend the proposed model to additional containerized services and integrate our proposed scheme into a container migration algorithm.

Acknowledgment

This work was partially sponsored by Brazilian National Council for Scientific and Technological Development (CNPq) grant nº 304990/2021-3.

References

[1] E. Viegas, A. Santin, J. Bachtold, D. Segalin, M. Stihler, A. Marcon, and C. Maziero, "Enhancing service maintainability by monitoring and auditing SLA in cloud computing," *Cluster Computing*, vol. 24, no. 3, pp. 1659–1674, Nov. 2020.

[2] H. Nemati and M. R. Dagenais, "VM processes state detection by hypervisor tracing," in *2018 Annual IEEE International Systems Conference (SysCon)*. IEEE, Apr. 2018.

[3] F. Ramos, E. Viegas, A. Santin, P. Horchulhack, R. R. dos Santos, and A. Espindola, "A machine learning model for detection of docker-based APP overbooking on kubernetes," in *ICC 2021 - IEEE International Conference on Communications*. IEEE, Jun. 2021.

[4] V. Abreu, A. O. Santin, E. K. Viegas, and V. V. Cogo, "Identity and access management for IoT in smart grid," in *Advanced Inf. Net. and App.* Springer International Publishing, 2020, pp. 1215–1226.

[5] B. B. Bulle, A. O. Santin, E. K. Viegas, and R. R. dos Santos, "A host-based intrusion detection model based on OS diversity for SCADA," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, Oct. 2020.

[6] N. Bashir, N. Deng, K. Rzadca, D. Irwin, S. Kodak, and R. Jnagal, "Take it to the limit," in *Proceedings of the Sixteenth European Conference on Computer Systems*. ACM, Apr. 2021.

[7] Q. Ali, D. Dunn, R. Garg, X. Lu, T. Muirhead, R. Taheri, and J. Zubb, "Performance of vsphere 6.7 scheduling options," VMWare Inc., White Paper, 04 2019. [Online]. Available: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/performance/scheduler-options-vsphere67u2-perf.pdf

[8] N. M. Calcavecchia, O. Biran, E. Hadad, and Y. Moatti, "Vm placement strategies for cloud scenarios," in *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 2012, pp. 852–859.

[9] E. Viegas, A. O. Santin, and V. A. Jr, "Machine learning intrusion detection in big data era: A multi-objective approach for longer model lifespans," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 366–376, Jan. 2021.

[10] Z. Kozhirbayev and R. O. Sinnott, "A performance comparison of container-based technologies for the cloud," *Future Generation Computer Systems*, vol. 68, pp. 175–182, Mar. 2017.

[11] D. Reis, B. Piedade, F. F. Correia, J. P. Dias, and A. Aguiar, "Developing docker and docker-compose specifications: A developers' survey," *IEEE Access*, vol. 10, pp. 2318–2329, 2022.

[12] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access*, pp. 976–996, 2019.

[13] S. Tata, M. Mohamed, T. Sakairi, N. Mandagere, O. Anya, and H. Ludwig, "rsla: A service level agreement language for cloud services," in *IEEE Int. Conf. on Cloud Computing (CLOUD)*, 2016, pp. 415–422.

[14] V. Podolskiy, M. Mayo, A. Koay, M. Gerndt, and P. Patros, "Maintaining slos of cloud-native applications via self-adaptive resource sharing," in *2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 2019, pp. 72–81.

[15] P. Ntambu and S. A. Adeshina, "Machine learning-based anomalies detection in cloud virtual machine resource usage," in *2021 1st International Conference on Multidisciplinary Engineering and Applied Science (ICMEAS)*, 2021, pp. 1–6.

[16] C.-H. Huang, B.-H. Huang, and T.-Y. Wu, "Hardware resource reliability analysis based on deep learning for virtual machine deployment optimization," in *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, 2020, pp. 726–727.

[17] S. Wang, Y. Yao, Y. Xiao, and H. Chen, "Dynamic resource prediction in cloud computing for complex system simulatiuon: A probabilistic approach using stacking ensemble learning," in *2020 International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI)*, 2020, pp. 198–201.

[18] D. Masouros, S. Xydis, and D. Soudris, "Rusty: Runtime interference-aware predictive monitoring for modern multi-tenant systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 184–198, 2021.

[19] P. Horchulhack, E. K. Viegas, and A. O. Santin, "Toward feasible machine learning model updates in network-based intrusion detection," *Computer Networks*, vol. 202, p. 108618, Jan. 2022.

[20] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *IEEE Int. Conf. on Data Eng. Workshops (ICDEW)*, 2010, pp. 41–51.

[21] E. K. Viegas, A. O. Santin, V. V. Cogo, and V. Abreu, "A reliable semi-supervised intrusion detection model: One year of network traffic anomalies," in *IEEE Int. Conf. on Comm. (ICC)*, 2020, pp. 1–6.