An Empirical Analysis of Common OCI Runtimes' Performance Isolation Capabilities

Simon Volpert
simon.volpert@uni-ulm.de
Ulm University
Institute of Information Resource Management
Ulm, Germany

Sascha Winkelhofer
sascha.winkelhofer@gini.net
Gini GmbH
Munich, Germany

Stefan Wesner
wesner@uni-koeln.de
University of Cologne
Cologne, Germany

Jörg Domaschka
joerg.domaschka@benchant.com
BenchANT GmbH
Ulm, Germany

ABSTRACT

Industry and academia have strong incentives to adopt virtualization technologies. Such technologies can reduce the total cost of ownership or facilitate business models like cloud computing. These options have recently grown significantly with the rise of Kubernetes and the OCI runtime specification. Both enabled virtualization technology vendors to easily integrate their solution into existing infrastructures, leading to increased adoption. Making a detailed decision on a technology selection based on objective characteristics is a complex task. This specifically includes the instrumentation of performance characteristics that are an important aspect for a fair comparison. Moreover, a subsequent quantification of the isolation capability based on performance metrics is not readily available.

In this paper, we instrument and determine the OCI runtime isolation capability by measuring virtualized system resources. We hereby build on two previous contributions, a proven isolation measurement workflow engine, and meaningful isolation metrics. The existing workflow engine is extended to integrate OCI runtime instrumentation as well as the novel isolation metrics.

We indicate a quantifiable distinction between the isolation capabilities of these technologies. Researchers and industry alike can use the results to make decisions on the adoption of virtualization technology based on their isolation characteristics. Furthermore, our extended measurement workflow engine can be leveraged to conduct further experiments with new technologies, metrics, and scenarios.

CCS CONCEPTS

• **General and reference → Performance**; • **Computing methodologies** → *Modeling methodologies.*

KEYWORDS

Isolation, Virtualization, Benchmarking, Framework, Container

ICPE '24, May 7–11, 2024, London, United Kingdom
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0444-4/24/05.
https://doi.org/10.1145/3629526.3645044

ACM Reference Format:
Simon Volpert, Sascha Winkelhofer, Stefan Wesner, and Jörg Domaschka. 2024. An Empirical Analysis of Common OCI Runtimes' Performance Isolation Capabilities. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3629526.3645044

1 INTRODUCTION

Virtualization technologies are consistently driving the vision of a software-defined infrastructure. Implementing virtualization is motivated by various factors, from facilitating business models like cloud computing to potentially reducing total ownership costs. Since the early days of Virtual Machines (VMs)[5], the landscape has been massively enriched by novel approaches such as containerization and other lightweight virtualization concepts. The rise and growing market share of the Kubernetes container orchestration engine, as well as the definition of the Open Container Initiative (OCI) specification, led to an increasing number of tools, methodologies, runtimes and engines in the domain of virtualization. This enabled virtualization technology vendors to implement their runtimes according to the OCI specification to be utilized interchangeably with orchestrators like Kubernetes [14]. For industry and academia alike, this vast number of options makes it difficult to objectively decide on what technology to utilize. The rapid expansion in this area makes it hard to stay abreast of all the most recent developments.

Comparing different virtualization technologies is a multidimensional decision problem with criteria ranging from security considerations, isolation capabilities, the type of virtualization, and many more. Various research studies are conducted on the comparison of virtualization technologies in several aspects, from impact on startup times[16], security considerations[15], to performance and isolation analysis[12].

Due to this mentioned rapid extension of the virtualization landscape, we need effective means to make informed decisions based on objective metrics. Multi-criteria decisions are typically complex and can hardly be reduced to a single metric [3]. Thus, we focus on the single distinct "isolation" metric to compare virtualization technologies against each other.

In systems with multiple competing workloads, isolation efficiency can be quantified by the impact that a disruptive workload has on its competing but behaving counterpart. Therefore, it is necessary to measure the performance characteristics of the different resources that are contended. To objectively analyze these characteristics for different runtimes, the instrumentation across all technologies has to be done from a black-box perspective. It needs to acquire similar performance metrics for all technologies and requires low instrumentation overhead. We take advantage of extended Berkeley Packet Filter (eBPF) to get an unobstructed view of the performance characteristics. For the sake of acquiring similar metrics for all technologies and simplification of implementation, we focus on OCI compliant virtualization runtimes.

In this work, we answer the following research questions.

RQ 1 (instrumentation). *How can performance and derived isolation characteristics of OCI compatible runtimes be instrumented and subsequently measured?*

RQ 2 (automatability). *How can the instrumentation across different OCI runtimes be conducted in an efficient and uniform manner?*

RQ 3 (comparision). *How do the isolation capabilities among OCI compatible virtualization technologies compare?*

This paper uses an existing benchmark-based evaluation methodology that supports the instrumentation of performance degradation and the determination of isolation capabilities. More precisely, we present the following contributions.

C 1 (isolation determination framework). *We release the codebase of the evaluation framework including the extensions developed during the work on this paper [20]. This relates to* **RQ 1** *and* **RQ 2** *and enables fellow researcher to perform similar measurements for their usecases.*

C 2 (comparison). *We present a comparison (***RQ 3***) of three distinct OCI compliant virtualization technologies regarding their isolation capabilities.*

The remainder of this paper is structured as follows. In section 2 we briefly present the fundamentals of this work. This includes eBPF, OCI and a discussion of isolation and its quantification. This is followed by a description of the methodology in section 3 and lays the foundation for the answer to **RQ 1**. The methodology is followed by some important details of the implementation in section 4. It discusses the remaining aspects of answering **RQ 1** and additionally answers **RQ 2**. Section 5 gives a brief overview of the technologies involved in the experimental setup. This setup is used to generate the final results in section 6 which closes **RQ 3**. We finish with a discussion in section 7, a review of related work in section 8 and a final summary in section 9.

2 BACKGROUND

This section explains fundamentals that are essential for the further course of this work.

2.1 eBPF and Instrumentation

This section briefly highlights eBPF and Linux profiling. A more detailed description is available in the previous work of the fellow authors [2, 21].

eBPF enables the execution of verified code within a special VM that runs as part of the Linux kernel. It hereby extends the capabilities of the initially developed Berkeley Packet Filter (BPF) developed at a Berkeley Laboratory [13]. Apart from executing functions when receiving network packets, it can now observe and react to a multitude of event sources as part of the Linux profiling subsystem. Those specifically include Performance Monitoring Counters (PMCs), tracepoints, kernel, and user functions.

While these events are technically not part of eBPF, it still enables an approachable exploitation of them. The typical lifecycle of an eBPF program is visualized in Figure 1 as presented by Gregg [6].

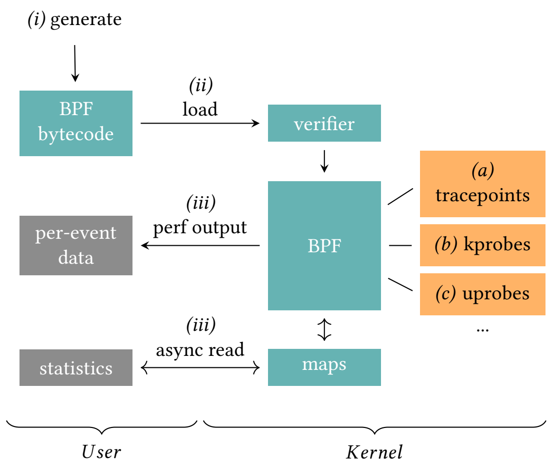


Figure 1: eBPF internals and Linux instrumentation according to [6]

The following paragraphs briefly describe the previously mentioned Linux profiling subsystem instrumentation points that are relevant to this work.

(a) *Tracepoints.* Tracepoints are static kernel instrumentation points[19]. They are defined and implemented by the kernel developers and issue an event once a specific call occurs. They further include counters that are specific to hardware, like CPU instrumentation.

(b) *Kprobes.* Kprobes are similar to tracepoints, yet not statically defined [7]. They allow dynamic hooks into any kernel function call. As this depends on the kernel function name, this is not stable across kernel releases.

(c) *Uprobes.* Similar to kprobes, uprobes can dynamically instrument user space function calls. In practice, this requires available debug symbols [4].

2.2 Open Container Runtime

The "OCI"[1] is part of the "Linux Foundation"[2] that develops open standards for container-based virtualization. These open standards take the form of specifications. In the context of this paper, the "runtime specification" is of particular importance. Figure 2 highlights

[1]https://opencontainers.org/
[2]https://linuxfoundation.org/

the complete container virtualization toolchain from user input to running the actual *(v)* container. Mavridis and Karatza [12] describe this structure and the accompanying technologies in detail.

A *(iv)* runtime that implements the *(iii)* OCI runtime specification can be utilized by *(ii)* container engines that provide a respective interface. Popular technologies that implement this interface include containerd[3], Podman[4] and CRI-O[5]. The engines offer an Application Programming Interface (API) that can be used by *(i)* user-experience-oriented container management and orchestration tools such as Docker[6] or Kubernetes[7].

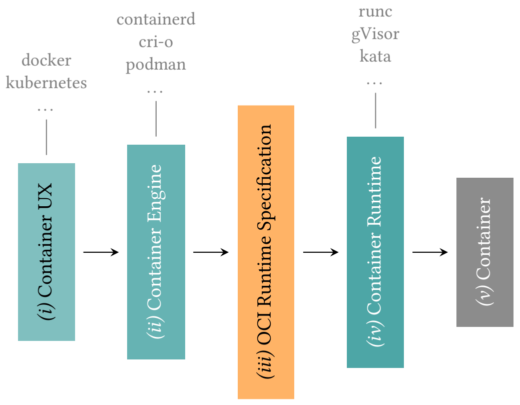


Figure 2: OCI ecosystem

As Figure 2 suggests, there are many combinations of tools within this chain possible, as each segment is individually interchangeable. Specifically, the widely adopted Kubernetes orchestrator created a large amount of the so-called "Kubernetes distributions" that bundle toolchains in an opinionated use case-driven manner. Popular examples are SUSE's[8] k3s[9] and RedHat's[10] OpenShift[11].

Another popular example for OCI compliant technology combinations is the container UX solution *Docker*. While docker initially started out as a full stack container solution including engine and runtime, it open sourced its components (containerd, runc) and now focusses on its role as management and orchestration tool.

2.3 Isolation Terminology

Isolation is a state that occurs when two workloads share and thus compete for a resource. The degree to which they influence each other describes isolation. If they have a strong impact on each other, the isolation is low and vice versa. [9, 11, 24]. This work follows the isolation definition of Krebs et al. [9] who define performance isolation as follows:

Definition 1 (Isolation). *Performance isolation is the ability of a system to ensure that tenants working within their assigned quota (i.e., abiding tenants) will not suffer performance degradation due to other tenants exceeding their quotas (i.e., disruptive tenants).*

In a similar context and especially in cloud computing, related work regularly uses the term "noisy neighbor". This noisy neighbor describes a disruptive tenant that negatively impacts another tenant. According to Longbottom [10], it is defined as follows:

Definition 2 (Noisy Neighbor). *A workload within a shared environment is utilizing one or more resources in a way that it impacts other workloads operating around it.*

2.4 Isolation Quantification

Throughout this work, we assume two distinct workloads W_a and W_d. The workloads themselves enact a certain amount of resource utilization.

W_a describes the abiding, behaving workload that stays within its assigned limits and utilizes a constant amount of resources. W_d on the other hand, defines a disruptive workload that misbehaves in one way or another. It may do so by actively trying to disturb other workloads as a "Noisy Neighbor" or by inadvertently negatively impacting other workloads due to an error.

A simple and natural approach to the quantification of isolation is the calculation of a "performance loss rate" [8, 11, 17, 24]. It describes the amount of performance degradation of W_a when affected by W_d on a fixed amount of workload.

Therefore, the baseline performance of a workload W_{a_1} in an uncontended environment is measured. Subsequently, the same workload plus an additional disrupting workload W_d is started, resulting in workload performance W_{a_2}. Both workloads compete against resources.

The isolation performance loss rate I_{plr} as the rate between the difference of both performance measurements can then be determined as shown in Equation (1). Slightly changing the perspective, I_{ulr} refers in Equation (2) to the utilization loss rate relative to the maximum possible utilization $R_{n_{max}}$ that a resource R_n can achieve.

$$I_{plr} = \frac{|W_{a_1} - W_{a_2}|}{W_{a_1}} \quad (1) \qquad\qquad I_{ulr} = \frac{|W_{a_1} - W_{a_2}|}{R_{n_{max}}} \quad (2)$$

In addition to this simplified model, further distinctions can be made. As part of his dissertation, Krebs et al. developed a model that included a graphical representation of isolation characteristics. It incorporates several interesting isolation points of interest along the range of values [9]. On the basis of those, additional metrics can be derived. Their graphical representation is adopted here and is presented in Figure 3.

In the following, we briefly iterate over their most notable metrics in the context of this paper.

Generally, this model also assumes static workload W_a. In contrast to the performance loss rate as described above, the disruptive workload W_d increases its load over time and as a consequence impacts W_a

The x-axis W_d in Figure 3 represents the amount of workload the disruptive tenant causes, whereas the y-axis W_a represents that for the abiding tenant.

[3]https://containerd.io/
[4]https://podman.io
[5]https://cri-o.io/
[6]https://docker.com/
[7]https://kubernetes.io/
[8]https://www.suse.com/
[9]https://k3s.io/
[10]https://www.redhat.com/
[11]https://www.openshift.com/

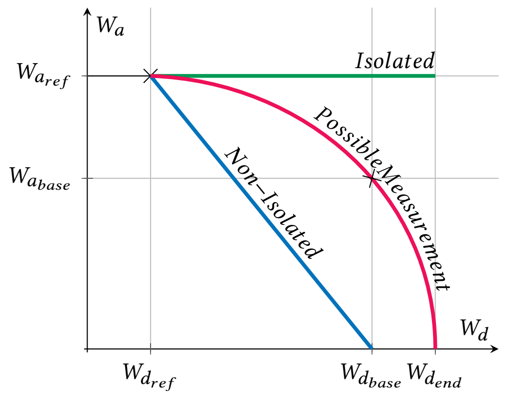


Figure 3: Visualization of theoretical and practical behaviour in isolation scenarios

The green line denoted with "isolated" shows a perfectly isolated W_a, which is not affected by W_d at all. No amount of W_d has an impact on W_a.

In contrast, the blue line indicated with "non-isolated", shows how workload W_a decreases, while workload W_d increases. For this case, no isolation occurs at all and W_d is clearly prioritized.

In reality, the actual graphical representation is represented by the red line denoted by "Possible Measurement". It will lie somewhere between the aforementioned green and blue lines.

With the graphical representation in Figure 3 in mind, various interesting isolation points can be identified. Moreover, these offer the potential to derive additional useful metrics. The following briefly iterates on the said points as defined by Krebs et al. [9].

Reference points. The reference point $W_{d_{ref}}$ marks where the disruptive tenant starts to degrade the abiding tenant. In consequence $W_{a_{ref}}$ defines this point from the perspective of W_a.

Degradation points. $W_{d_{base}}$ highlights the point where W_a is fully degraded and thus reduced to zero if W_a was not isolated. The same is true for $W_{d_{end}}$, marking the point where an arbitrary isolation would cause that degradation to zero. Finally, the cross section of $W_{a_{base}}$ and $W_{d_{base}}$ marks the respective workload for that arbitrary isolation that would otherwise be zero if no isolation had taken place.

Krebs et al. give an example for an isolation metric based on degradation points. They describe the difference between $W_{d_{end}}$ and $W_{d_{base}}$ in relation to $W_{a_{ref}}$. This relation is shown as I_{end} in Equation (3). Its value is zero if no isolation happens, and the higher this value gets, the better the isolation is. As this value tends to ∞, the authors suggest to rather use $W_{a_{ref}}$ as a reference resulting in values between $[0, 1]$. This is in Equation (4).

$$I_{end} = \frac{W_{d_{end}} - W_{d_{base}}}{W_{a_{ref}}} \quad (3) \qquad I_{base} = \frac{W_{a_{base}}}{W_{a_{ref}}} \quad (4)$$

The authors further argue that these metrics are sufficient only for systems that degrade linearly. They therefore propose two additional integral-based metrics. Equation (5) describes the relation between the area under the measured curve and the area under the curve of the non-isolated workload starting from $W_{d_{ref}}$. Equation (6) describes the same relation but starting from $W_{d_{base}}$ to an arbitrary point beyond $W_{d_{base}}$. The latter could be the highest value on $W_{d_{base}}$. Both values range between $[0, 1]$.

$$I_{intBase} = \frac{\left(\int_{W_{d_{ref}}}^{W_{d_{base}}} f_m(W_d)dW_d\right) - W_{a_{ref}}^2/2}{W_{a_{ref}}^2/2} \quad (5)$$

$$I_{intFree} = \frac{\left(\int_{W_{d_{ref}}}^{p_{end}} f_m(W_d)dW_d\right) - W_{a_{ref}}^2/2}{W_{a_{ref}}^2 \cdot \left(p_{end} - W_{d_{ref}}\right) - W_{a_{ref}}^2/2} \quad (6)$$

Not every metric described here will ultimately be useful for this work. Although we will elaborate on their applicability in section 4.

3 METHOD

This section presents the underlying method applied to gather isolation metrics. It starts by briefly summarizing this work's goal and follows by discussing the applied scenarios, instrumentation, and isolation quantification methods.

3.1 Goals

To reiterate our research questions, our aim is to measure the isolation of certain OCI based virtualization technologies. Therefore, we isolate them against each other in specific scenarios that are presented in the upcoming section. To achieve this, we instrument them directly on the host as in outside their virtualization environment. This enables us to collect high-resolution performance metrics of any involved process. This instrumentation is possible by leveraging eBPF. The actual isolation quantification follows related work. We decide to determine different metrics in order to compare and discuss them in conclusion.

3.2 Scenarios

We analyze the characteristics of isolation among tenants in four distinct scenarios. Each scenario consists of an abiding and a disruptive workload according to the model presented by Krebs et al. [9]. The abiding workload statically utilizes a resource and is contended by a disruptive workload. This disruptive workload continuously increases its workload until it reaches its final utilization. An overview of all scenarios is presented in Table 1. In this table, l_a and l_d describe the resource limits for w_a and w_d compared to the total available resources. w_a and w_d describe the relative utilization of the respective workloads within these limits.

For each scenario, the abiding workload utilizes a static amount of resources, whereas the disruptive workload increases linearly over time. This enables us to measure every possible combination of workloads between a static abiding and a linear disruptive workload. Plotting both on distinct axes results in a graph similar to Figure 3.

The runtime of each scenario is sensibly chosen depending on the maximum degree of utilization, its maximum capacity, and the time it takes to utilize it. The linear increasing disruptive workload

Table 1: Isolation scenarios with workload and their limits

name	w_a	l_a	w_d	l_d
harmony	100 %	50 %	100 %	50 %
escape	100 %	50 %	150 %	50 %
overcommitting	100 %	50 %	100 %	75 %
steal	50 %	50 %	100 %	75 %

evenly utilizes its resource over the experiment runtime and thus subdivides into fitting steps whose size and degree of isolation depend on the same characteristics as the total runtime. The experiments themselves are each repeated at least 10 times and the physical server involved is fully reset in between to mitigate any residue and thus impact from previous experiment runs [18].

harmony. In this scenario, two workloads fully utilize their assigned resources. The assigned resources are imposed through limits and evenly distributed throughout the resource as a whole, resulting in 50% for both. In practice, this means that the capacity planning performed previously adhered strictly to the combined available resources. This is a typical use-case for scenarios where no overcommitting or dynamic resource sharing happens. From a theoretical point of view, no resource contention should occur, and thus both workloads should not interfere with each other.

escape. This scenario is similar to the harmony one with only one exception. Here, the disruptive workload tries to escape its own imposed limit. Such scenarios occur when a workload accidentally or on purpose tries to exceed its allocated resources. In consequence, we can see two different things. One being whether the limit can actually be imposed and the workload is not able to exceed its limit, and the other being whether it is able to have an impact on the abiding workload in either case. In an ideal case, there should be no impact whatsoever.

overcommitting. Overcommitting is something that typically happens in cloud scenarios. As briefly mentioned in section 1 this may be part of their business model. In this scenario, we set a higher total limit for a resource than is physically available. To consider the worst case, both workloads use 100% of resources within their own limit. Again, considering the cloud computing use case, this most likely leads to a violation of customers Service Level Agreement (SLA). As both workloads stay within their limits, it is not deterministic how the workloads will behave. Nevertheless, it is still interesting how degradation occurs and whether different virtualization technologies behave differently.

steal. This is an extension of the "overcommitting" scenario. Although the limits still exceed the total available resource, workloads no longer use them fully. The abiding workload purposefully utilizes only half of its granted resources, while the disruptive still fully uses the resource until its limit. The combined workloads fully utilize all available resources, however. Essentially, this scenario tests whether the disruptive workload can steal free available resources from the abiding one. Some virtualization technologies may allow this, whereas others may strictly assign resources and block them.

3.3 Instrumentation

The method for resource instrumentation is based on the principles designed by [21]. In summary, this implies that the instrumentation must follow two essential functions. It *(i)* must occur outside of the virtualization technology to get a holistic view of the unobstructed resources. Furthermore, *(ii)* needs to be independent of virtualization technology to allow a fair comparison. This means that the instrumentation points need to be reasonably similar. Therefore, the uprobes as outlined in section 2.1 are not applicable.

eBPF has made it possible to access any kind of Linux instrumentation while promising low instrumentation overhead. The overhead can be kept low, as it can be performed within the kernel, reducing the amount of overhead induced by frequent kernel userspace interactions significantly.

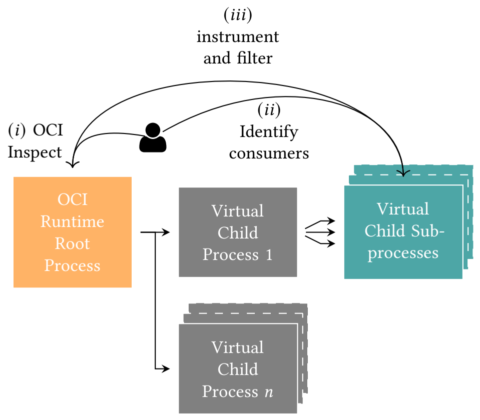


Figure 4: Instrumentation of processes controlled by OCI runtimes

As the instrumentation happens outside the virtualization technology, the following issues need to be solved. The process to do is highlighted in Figure 4.

We first need to *(i)* identify and distinguish virtualization technology instances that compete for resources. We further need to *(ii)* identify the actual process that consumes the resources. Finally, we need to *(iii)* find an appropriate instrumentation point that correctly profiles the resource in question.

When it comes to network resource profiling, the eXpress Data Path (XDP) feature of eBPF offers an efficient way of instrumenting. XDP provides an API to implement functions that are attached directly to network interfaces and allow stateless processing of incoming packets. It promises fast networking functionality, as it allows bypassing the Linux netfilter stack. State handling, including, but not limited to, packet counting and stateful connection tracking, can be achieved by leveraging eBPF maps that can be accessed by the named XDP function, as well as by user-space applications.

3.4 Isolation Quantification

In section 2.4 we presented and briefly discussed several useful metrics to quantify isolation. However, not all are desirable or even applicable to our scenarios as shown in section 3.2.

For example, I_{plr} in Equation (1) is widely applied in scientific work, but that may also be due to it being straightforward and comparatively simple. For these reasons and to compare it with another metric, we still calculate it. In order to do so, we need to pick a performance degradation point in an isolation diagram like Figure 3. Here, we choose the performance degradation where the disruptive workload is at its maximum. Another viable option would be the highest degradation of the abiding workload observed. However, we did not observe a significant difference between these possible points and therefore neglected them.

In section 2.4 we cite Krebs et al. [9] who argues that metrics like I_{plr} are only sufficient for linearly degrading resources. However, this is not always the case, as we can see in our results in section 6. They therefore suggested integral-based ones. However, some metrics assume that we have a disruptive workload that is capable of fully degrading the abiding one. Our scenarios do not force that, and depending on the resource, the respective resource scheduler might not allow this. As a consequence, we decide to use the $I_{int_{free}}$ metric in Equation (6). This metrics calculation only regards the first point of degradation and the highest applied disruptive workload.

In section 7 we reiterate the correlations for these metrics and discuss their respective applicability retrospectively. We therefore introduce a simple isolation similarity metric S_I between $I_{int_{free}}$ and I_{plr} that describes how similar they are to each other on a scale from $[0, 1]$ where 1 is exactly the same and 0 very different.

4 IMPLEMENTATION

The evaluation process is based on the evaluation framework presented in [21]. Some aspects are extended to enable answering the research questions. In particular, this involves updates to the load generation and instrumentation details.

Compared to the initial work, the three notable changes are *(i)* new scenarios as presented in section 3.2, *(ii)* new isolation metrics as described in section 3.4 and new technologies as shown in section 5.2.

4.1 Workflow

The whole process of load generation and data acquisition for every possible combination of scenario and technology relies on the evaluation framework. The abstract process of a conducted experiment is briefly highlighted below.

It follows the workflow highlighted in Figure 5. Here, the process begins with the *(i)* spawning of a virtualization technology process. Within this *(ii)* load is generated by the respective load generation tools. Afterwards, the *(iii)* profiling process on the host system is started in parallel. This profiling supervises and profiles the virtualization technology process. Upon success, data is *(iv)* acquired and *(v)* stored on external storage.

4.2 Load generation

The original load generation process needs several changes in order to enact the previously mentioned scenarios. Specifically, the static load generation at several distinct interesting points is changed to a linear load generation phase. This improvement still includes all previous configurations and extends them with a configurable number of points.

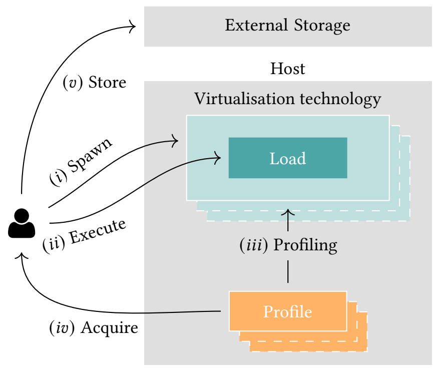


Figure 5: Flow of an abstract measurement

Benchmarking and load generation tools that are scientifically trusted do not commonly offer the possibility to gradually increase the load over given amount of. We therefore separate the load generation into multiple intervals with configurable resolution. By doing so, we can achieve a nearly linear behavior of load generation, as highlighted in Figure 6.

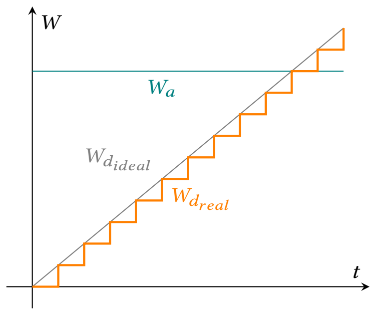


Figure 6: Visualization of linear load generation highlighting ideal and real W_d

Here, the axis describes the workload over time showing the abiding constant workload W_a as well as the theoretically ideal disruptive workload $W_{d_{ideal}}$ and the actual real disruptive workload $W_{d_{real}}$.

4.3 Instrumentation

In practice, the process structures of distinct virtualization technologies are very different. Adding a new technology to this workflow engine would involve a small implementation effort based on process structure investigations. To give an example, the process structure of loosely isolated container technology like docker with runc is visible in detail, whereas it is mostly abstracted for hypervisor-based virtualization like KVM. In the latter case, we cannot see processes running within the virtual machine from the outside.

The following Table 2 presents all the instrumentation points for the technologies observed in this work and the process filter necessary to sort the process trees of competing tenants. We specifically avoided kprobes, as they are significantly less stable than

instrumentation	resource	description
hardware:cycles	CPU	Hardware counter reporting CPU cycles
tracepoint:kmem:rss_stat	Memory	Tracepoint called when Resident Set Size (RSS) counters change
tracepoint:block:block_io_start	Disk	Tracepoint called when block operation request is queued for execution
XDP	Network	Network interface specific functions executed upon incoming network packet

Table 2: Instrumentation points list

name	version	note
Fedora CoreOS	39	Operating system version
Linux Kernel	6.5.6	Kernel used by the operating system Fedora CoreOS
k3s	v1.28.4	Rancher Kubernetes Distribution
Argo Workflow	v3.5.0	The workflow engine to orchestrate experiments and scenarios
bpftrace	v0.19.0	Profiling tool based on eBPF and bcc
stress-ng	0.13.05	Load generator for CPU and memory
fio	3.28	Load generator for disk I/O
iperf3	3.10.1	Load generator for network I/O
podman	4.7.0	
gvisor	20231023	with the systap platform
kata	3.2.0	with the virtio-fs storage driver

Table 3: Software version list

tracepoints. This is no hard requirement, though, and extensions of this framework might make their usage necessary.

To measure the throughput on a network interface, we implement an XDP function that increments time-based buckets in an eBPF-map. After a finished run, all buckets are then extracted by a user-space application for further analysis. The OCI specification dictates the use of veth-pairs to be OCI compliant. This allows us to perform black-box measurements of network isolation capabilities, independent of the OCI runtime under test.

5 EXPERIMENTAL SET-UP

5.1 Physical nodes

The experimental setup consists of 7 physical servers. They are arranged symmetrically and consist of identical components. The CPUs are two Intel CPUs of the model "Intel(R) Xeon(R) CPU E5-2630 v3" with a basic clock frequency of 2.40 GHz and a maximum clock frequency of 3.20 GHz. The memory attached to those CPUs have a total of $16 \cdot 16 = 256$ GiB DDR4 memory clocked at 2133 MHz available. The disk involved at the Input Output Operations Per Second (IOPS) isolation tests is a Samsung SM843TN, rated with 15000 IOPS "random write" performance. The server types involved are six experiment nodes for parallel execution and one control node that provides bare metal provisioning for the workflow control engine.

The networking between all involved nodes is realized by Mellanox Technologies Network Interface Card (NIC) of the "MT27800 ConnectX-"5 family. These are capable of a network throughput rate of $50Gbit/s$. Nevertheless, as described during the results in section 6.4 they are not used for the actual network resource experiments. Here we use on-board network cards that only provide a maximum throughput of $1Gbit/s$, as otherwise there would be no resource contention for hypervisors that do not offer bandwidths beyond a few $Gbit/s$

Notable other software components involved are listed in Table 3. All of these are part of the automated experiment workflow engine as described in section 4.1.

5.2 Selected Virtualization Technologies

Based on the components deconstructed of virtualization technologies, we can roughly separate them into categories [21]. Although there are not necessarily strict categories for their isolation capabilities, they are sufficient for a rough starting point. Thus, for this work we select three popular engines including OCI compatible container runtimes that each fit into those proposed categories. More specifically, the technologies are Podman[12], gVisor[13] and Kata[14]. Figure 7 shows a hierarchical representation of those choices.

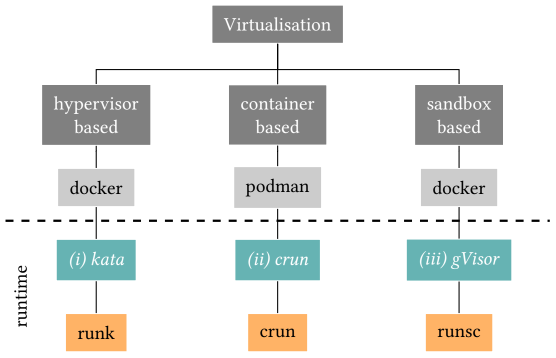


Figure 7: Virtualisation Classification Overview

Naturally, these technologies offer many possible configurations. In this paper, we use their most recent releases as of date, as well as the upstream default configuration. Table 3 gives an overview over these details in the bottom part. Here, we distinguish between the container runtime as defined in Figure 2 and the container runtime's binary name for reference.

crun. This container runtime, developed by RedHat, adheres to a more conventional approach. It leverages namespaces and cgroups for isolation, offering an alternative to runc, often bundled with containerd. Notably, it does not differentiate between the runtime and binary name, hence also referred to as "crun".

[12]https://podman.io/
[13]https://gvisor.dev/
[14]https://katacontainers.io/

***gVisor*.** Originated at Google[15] gVisor promises a stricter isolation among workloads. Although this technology also builds on namespaces and cgroups, they further improve isolation by filtering Linux system calls [26]. This "sandbox" approach reimplements fundamental Linux capabilities within the user space to gain more control and thus improve isolation [22]. The runtime "runsc" is bundled with gVisor itself.

***Kata*.** This independent OpenSource engine leverages hypervisor based virtualization to achieve isolation. Moreover, it also uses cgroups and namespaces where applicable. Similarly to the technologies mentioned above, Kata also bundles its container runtime "runk".

6 RESULTS

This section presents and discusses the results of the isolation measurements. Therefore, we present an overview in Table 4 that contains all isolation metrics determined based on the measurements we performed.

Table 4 consists of two multi-indexes. The vertical ones describe all permutations of virtualization technologies as described in section 5.2 and the scenarios we presented in section 3.2. Horizontal indices describe all permutations of resources that are instrumented as part of the workflow in section 4.1 and all isolation metrics discussed in section 3.4.

The number of results is too numerous to discuss every isolation characteristic in detail. As a consequence, we select interesting aspects for every resource and discuss them in the following sections. One thing all technologies have in common is that they allow granting unused resources to other workloads. In every steal scenario, the disruptive workload is able to allocate resources that could have been exclusively granted to the abiding one, without negatively impacting it. For this reason, we do not discuss this scenario further in the detail sections below.

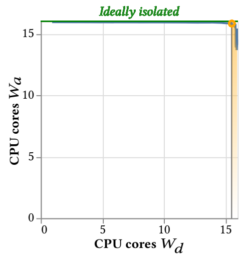
The figures "Isolation" and "Timeline" presented throughout the remainder of this section are aligned to Figure 3 and Figure 6, respectively. The isolation figures furthermore highlight the changepoint discussed in section 3.4 with an orange circle and highlight the area under the abiding but disturbed workload. This tries to give a better idea of what the $I_{int_{free}}$ metric will look like, since this is part of Equation (6).

6.1 CPU

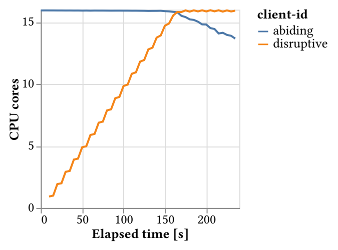
Reviewing the isolation metrics in the CPU column of Table 4, it is evident that CPU isolation is comparatively good and stable across the different virtualization technologies.

Considering a typical scenario like the escape one for the gVisor CPU, we can see that there is a slight degradation visible. However, S_I indicates that our two isolation models slightly disagree. To understand the reasoning behind this, we investigate the time and isolation charts.

According to the escape scenario, both workloads fully utilize the CPU together. In fig. 8b we can see the steadily increasing workload W of the disruptive workload until it tries to escape its limit. We can see that this is not possible. However, as soon as the disruptive

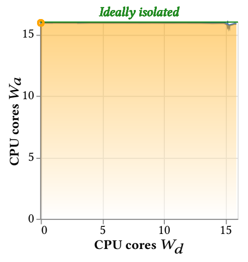


(a) Isolation

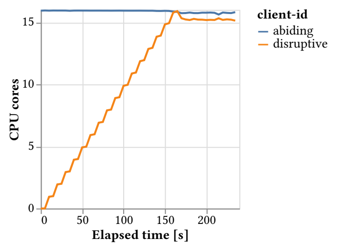


(b) Timeline

Figure 8: gVisor CPU Escape Scenario



(a) Isolation



(b) Timeline

Figure 9: Kata CPU Escape Scenario

workload reaches its limit, the abiding workload starts to degrade. This might be due to the fact that the CPU scheduler needs to take efforts to keep the disruptive workload from exceeding its limit, implying less available CPU time for the abiding workload.

In contrast, the same scenario for the Kata CPU isolation looks slightly different. In fig. 9b we cannot observe a significant degradation over the course of the experiment. The fact that there is no change point above our threshold detected in fig. 9a, has a huge effect on the areas under the graphs of the calculated $I_{int_{free}}$. This can be neglected, though, as these cut each other out by building a ratio as highlighted in Equation (6).

6.2 Disk

As visible in Table 4 it is imminent that disk isolation has problems for our scenarios. An immediately visible aspect is the fact that disk isolation for gVisor is not present. This is because it, at the time of writing, does not support directly passing block devices into its container. This is a mandatory requirement as we generate and measure direct block operations.

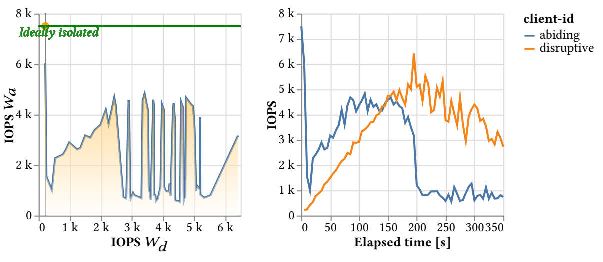
Apart from that observation, the disk isolation for the remaining technologies is arguably bad. Considering Podman, for example, yields bad results in almost every scenario with $I_{int_{free}} < 0.5$ and I_{plr} even worse.

This is clearly reflected in the respective time and isolation charts in Figure 10, taking the overcommitting scenario as an example. Here we can see that as soon as the disruptive workload starts, the abiding is almost reduced to zero. Interestingly, it starts to regain its workload over time, which will quickly be impacted by the ever-increasing disruptive workload, though. This behavior is a good example of how changing or specifically increasing the workload

[15]https://www.google.com/

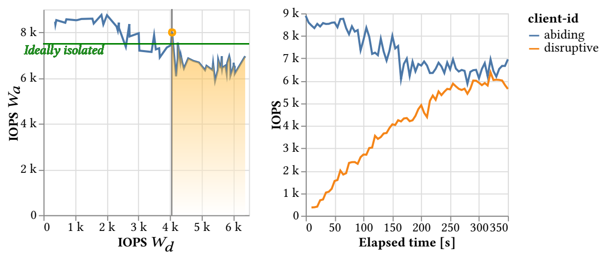
Table 4: Isolation metrics comparison

	Resource	cpu			disk			memory			network		
	Metric	$I_{intFree}$	I_{plr}	S_I	$I_{intFree}$	I_{plr}	S_I	$I_{intFree}$	I_{plr}	S_I	$I_{intFree}$	I_{plr}	S_I
Technology	Scenario												
gvisor	escape	0.89	0.85	0.96	n/a	n/a	n/a	0.99	0.99	1.00	n/a	n/a	n/a
	harmony	0.99	0.99	1.00	n/a	n/a	n/a	0.99	0.99	1.00	1.00	1.00	1.00
	overcommit	0.94	0.89	0.95	n/a	n/a	n/a	n/a	n/a	n/a	0.76	0.87	0.87
	steal	1.00	0.99	1.00	n/a	n/a	n/a	1.03	1.00	0.97	0.95	1.00	0.95
kata	escape	0.95	0.99	0.96	0.86	0.95	0.90	0.99	1.02	0.97	n/a	n/a	n/a
	harmony	0.99	0.99	1.00	0.60	0.93	0.64	1.00	1.02	0.99	1.00	1.00	1.00
	overcommit	0.88	0.80	0.92	0.44	1.04	0.43	n/a	n/a	n/a	0.96	0.94	0.98
	steal	1.00	1.00	1.00	1.42	1.31	0.92	1.10	1.16	0.94	1.00	0.99	0.99
podman	escape	0.94	0.84	0.89	0.41	0.13	0.32	0.93	0.91	0.97	n/a	n/a	n/a
	harmony	1.00	0.99	1.00	0.47	0.15	0.32	0.93	0.92	0.98	1.00	1.00	1.00
	overcommit	0.94	0.90	0.96	0.39	0.10	0.25	n/a	n/a	n/a	0.93	0.86	0.93
	steal	1.00	1.00	1.00	0.41	0.95	0.43	1.06	1.00	0.95	1.00	1.00	1.00



(a) Isolation (b) Timeline

Figure 10: Podman disk overcommit Scenario



(a) Isolation (b) Timeline

Figure 11: Kata disk harmony Scenario

has a different impact than two static competing workloads. This also relates to the comparative low S_I scores. As mentioned before, the less linear the degradation process is, the less applicable the I_{plr} metric becomes. For this metric, we only consider the highest applied disruptive workload that is not stable, leading to a I_{plr} with very low expressiveness.

However, in general, we can see that the contended resource is not actually the observed and limited IOPS but a related resource that is saturated. This naturally depends on many factors such as the physical type of the disk (e.g. HDD, SSD, etc.), the bus it is attached to (e.g. PCIE, SATA, SAS) or hardware specific details like the installed disk controller. The analysis of what exactly happens here is beyond the scope of this work and is left for future investigations.

The Kata disk isolation issue is very different. Although it is capable of effectively limiting disk IOPS for processes running inside the virtual machine, this does not include the disk IOPS performed by actual virtualization technology. The kind of hypervisor Kata uses does not offer the possibility to instrument the individual processes inside it through eBPF. Instrumenting the hypervisor in consequence adds IOPS it executes for its own overhead. This behavior can be seen in Figure 11 for the harmony scenario.

Although this limits the significance of these specific results, we were still able to observe performance degradation between the abiding and disruptive workload. However, we only consider workload below the actual targeted utilization as described in the scenarios in Table 1, to ensure that an actual degradation of the workload inside occurs.

6.3 Memory

The isolation results for the memory metrics come with a small limitation. During the benchmarks, we determined that the Kata runtime has issues with the Non-Uniform Memory Access (NUMA) architecture of our physical servers and was not properly accessing memory across the CPU boundaries. Therefore, we limited virtualization technology access to a specific node and performed the experiments there while only allocating the memory (half of total) attached to it.

Moreover, we did not conduct any overcommit scenarios, as overcommitting memory was not possible in general. The Out Of Memory (OOM) killer would quickly kill the processes involved in our experiment, rendering the results useless. overcommitting the actual allocated memory (as in Resident Set Size (RSS)) is not possible in our configuration.

As memory isolation is very similar across technologies, we briefly discuss Podmans' escape scenario as an example in Figure 12.

One notable aspect is the observation that the actual workload never fully utilizes its designated utilization. This is due to the fact

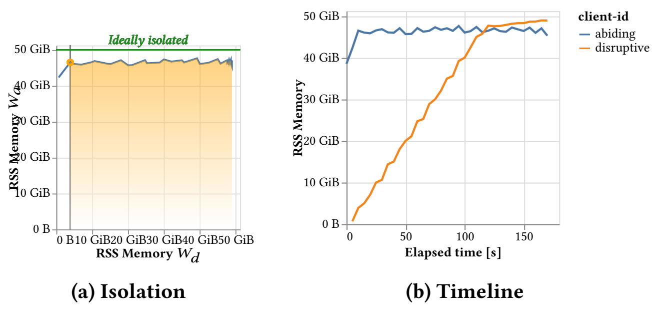


Figure 12: Podman memory Escape Scenario

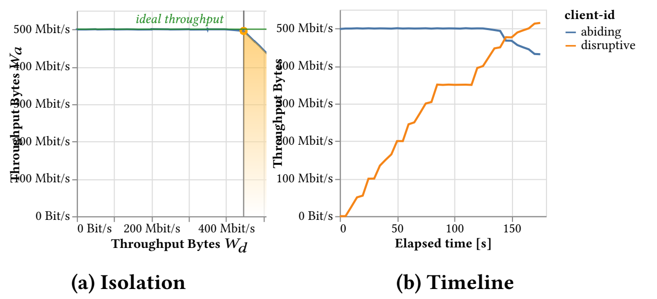


Figure 13: Podman network overcommit Scenario

that the load generator tries to allocate RSS memory with many workers in parallel and thus eventually reaches the limit. Once that happens, a process within the container gets OOM killed. One consideration here is that due to this circumstance, $I_{int_{free}}$ is lower than expected.

From the Figure 12 it is clearly visible that the imposed limitation for disruptive workloads works very well and cannot disturb the prevailing workload in any way.

6.4 Network

The virtualization technologies studied in this work do not offer the possibility of limiting the network bandwidth or IOPS of a NIC. This is the reason we do not perform an escape scenario for this resource, as it would merely mimic the overcommit scenario. However, technologies still possibly implement different strategies to enable network virtualization.

One significant difference is the Kata-induced limitation. Here, the maximum possible bandwidth achievable by the virtual NICs it creates is $5Gbit/s$. We therefore performed all our tests on an exclusive $1Gbit/s$ card specific for these experiments.

Apart from this consideration, the isolation metrics across all virtualization technologies are very similar. We see that as soon as both workloads fully utilize the NIC, performance degradation occurs. This behavior is clearly visible from any overcommit scenario as shown in Figure 13.

7 DISCUSSION

Although we can quantify the isolation capabilities of OCI runtimes by applying two different approaches to acquire isolation metrics, the results need critical reflection. This especially applies to the meaningfulness of the quantification methods in regard to the examined technology, scenario and resource, as well as the deviation of some results exceeding the predicted numeric range with an upper bound of 1.

As the respective S_I columns in Table 4 show, the deviation of the retrieved isolation metrics can be rather high across the technologies, applied scenarios, but also resources of interest. In section 3.4 we noted that I_{plr} is only sufficient for systems that degrade linearly. Looking at the corresponding graphs, it is evident that a linear degradation behavior does not apply to disk measurements. For some CPU measurements a near-linear behavior can be observed.

On the other hand, the integral-based metrics can also be misleading. As we calculate $I_{int_{free}}$ with the highest value of W_d, the applied scenario and therefore the maximum resource consumption of the disruptive tenant has a huge impact on the result of this metric.

In summary, depending on the selected configuration, the method to derive the isolation metric has to be selected carefully, the most expressiveness regarding the isolation capabilities is given by the isolation graphs.

Some calculated metrics show values higher than the theoretical upper bound of 1. For runtimes that hide information about running processes, such as gVisor, the retrieved values include the overhead produced by the runtime itself. A good example is the memory consumption measured for Kata containers. In each scenario, the retrieved values lie slightly above the imposed limits. Although Kata seems to apply the correct limits to the processes running inside, our method of measuring the consumption does not resemble the additional overhead of runtime. Subtracting overhead, to acquire more reasonable results, requires further investigation.

Generally speaking, the results that we present in this work are naturally very specific to the hardware, system configurations, and scenarios used. Isolation metrics cannot be easily compared between resources and scenarios. However, they can be compared within scenarios.

8 RELATED WORK

A common approach to quantify the isolation capacity of a virtualization technology is to determine the I_{plr} similar to Equation (1). Therefore, related work typically first measures a resource from within the virtualized environment[11, 17, 24, 25]. Combined with a subsequent measurement of the same workload under the influence of a disrupting contending workload, this ratio can be calculated. This approach has several constraints. One *(i)* is the measurement from within the virtualized environment. This neglects unforeseen impact on the host system. Another *(ii)* one is the dependability on the load generator specific to the stressed resource. Moreover, applying *(iii)* static workloads neglects effects induced by the variability of stress. Calculating only a *(iv)* single metric disregards the time in which a possible equilibrium of workloads is reached. In our work, we instrument the virtual environments from the hosting system through eBPF and thus decouple ourselves. Additionally, we induce a variable stress based on multiple scenarios and determine an isolation metric that takes the timeline into account.

This work focuses on the isolation of compute resources. Another aspect to consider is the impact that virtualization technology has on kernel resources and to classify isolation based on that. Wang et al. [23] follow very interesting approach to determine a misbehaving workload and further presents a tool to improve this isolation. Similarly Anjali et al. [1] assess virtualization technology categories along the kind and amount of system calls, and thus kernel stress.

9 CONCLUSION

Throughout this work, we presented a workflow engine and implementation details on isolation characteristics instrumentation for OCI compatible virtualization runtimes and give a rough estimate of the capabilities state-of-the-art virtualization technologies bring. Our results can be used in a decision process to pick a fit-for-purpose technology. Moreover, the framework itself can be extended for custom changes and executed on custom platforms.

We discovered some limitations in instrumentation and isolation for certain scenarios and technology combinations. These findings can be used to decide against a certain technology or to implement improvements.

One future direction that we intend to pursue with our work is to create a system of continuous profiling of isolation characteristics. Similarly to the work of Wang et al. [23], a classification of abiding and disruptive workloads could be based on a combination of performance metrics, derived isolation metrics, and kernel resource utilization. Furthermore, the characterization of workloads based on these metrics could be improved by integrating Quality of Service (QoS) metrics. Experiments may show a possible relation between resource isolation degradation and, for example, application response time.

REFERENCES

[1] Anjali, Tyler Caraza-Harter, and Michael M. Swift. 2020. Blending Containers and Virtual Machines: A Study of Firecracker and gVisor. In *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '20)*. Association for Computing Machinery, New York, NY, USA, 101–113. https://doi.org/10.1145/3381052.3381315

[2] Jörg Domaschka, Simon Volpert, Kevin Maier, Georg Eisenhart, and Daniel Seybold. 2023. Using eBPF for Database Workload Tracing: An Explorative Study. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. ACM, Coimbra Portugal, 311–317. https://doi.org/10.1145/3578245.3584313

[3] Jörg Domaschka, Simon Volpert, and Daniel Seybold. 2020. Hathi: An MCDM-based Approach to Capacity Planning for Cloud-hosted DBMS. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. 143–154. https://doi.org/10.1109/UCC48980.2020.00033

[4] Srikar Dronamraju. [n. d.]. *Uprobe-tracer: Uprobe-based Event Tracing — The Linux Kernel documentation.* https://docs.kernel.org/trace/uprobetracer.html Accessed: 2023-02-27.

[5] Robert P. Goldberg. 1973. *Architectural Principles for Virtual Computer Systems*. Technical Report. HARVARD UNIV CAMBRIDGE MA DIV OF ENGINEERING AND APPLIED PHYSICS. https://apps.dtic.mil/sti/citations/AD0772809 Section: Technical Reports.

[6] Brendan Gregg. 2017. Linux eBPF Tracing Tools. https://www.brendangregg.com/ebpf.html.

[7] Masami Hiramatsu. [n. d.]. Kprobe-based Event Tracing — The Linux Kernel documentation. https://docs.kernel.org/trace/kprobetrace.html Accessed: 2023-02-27.

[8] Samuel Kounev, Klaus-Dieter Lange, and Jóakim von Kistowski. 2020. *Systems Benchmarking: For Scientists and Engineers.* Springer International Publishing, Cham. https://doi.org/10.1007/978-3-030-41705-5

[9] Rouven Krebs, Christof Momm, and Samuel Kounev. 2012. Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments. In *Proceedings of the 8th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA '12)*. Association for Computing Machinery, New York, NY, USA, 91–100. https://doi.org/10.1145/2304696.2304713

[10] Clive Longbottom. 2017. *The Evolution of Cloud Computing: How to Plan for Change.* BCS Learning & Development Ltd, Swindon, UK.

[11] Jeanna Neefe Matthews, Wenjin Hu, Madhujith Hapuarachchi, Todd Deshane, Demetrios Dimatos, Gary Hamilton, Michael McCabe, and James Owens. 2007. Quantifying the Performance Isolation Properties of Virtualization Systems. In *Proceedings of the 2007 Workshop on Experimental Computer Science - ExpCS '07*. ACM Press, San Diego, California, 6–es. https://doi.org/10.1145/1281700.1281706

[12] Ilias Mavridis and Helen Karatza. 2023. Orchestrated Sandboxed Containers, Unikernels, and Virtual Machines for Isolation-Enhanced Multitenant Workloads and Serverless Computing in Cloud. *Concurrency and Computation: Practice and Experience* 35, 11 (2023), e6365. https://doi.org/10.1002/cpe.6365

[13] Steven McCanne and Van Jacobson. 1993. The BSD Packet Filter: A New Architecture for User-Level Packet Capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings (USENIX'93)*. USENIX Association, USA, 2.

[14] Marek Moravcik, Martin Kontsek, Pavel Segec, and David Cymbalak. 2022. Kubernetes - Evolution of Virtualization. In *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. 454–459. https://doi.org/10.1109/ICETA57911.2022.9974681

[15] David Schrammel, Samuel Weiser, Stefan Mangard, and Richard Sadek. 2022. Jenny: Securing Syscalls for PKU-based Memory Isolation Systems. (2022), 18.

[16] Martin Straesser, André Bauer, Robert Leppich, Nikolas Herbst, Kyle Chard, Ian Foster, and Samuel Kounev. 2023. *An Empirical Study of Container Image Configurations and Their Impact on Start Times.* https://doi.org/10.1109/CCGrid57682.2023.00019

[17] Xuehai Tang, Zhang Zhang, Min Wang, Yifang Wang, Qingqing Feng, and Jizhong Han. 2014. Performance Evaluation of Light-Weighted Virtualization for PaaS in Clouds. In *Algorithms and Architectures for Parallel Processing (Lecture Notes in Computer Science)*, Xian-he Sun, Wenyu Qu, Ivan Stojmenovic, Wanlei Zhou, Zhiyang Li, Hua Guo, Geyong Min, Tingting Yang, Yulei Wu, and Lei Liu (Eds.). Springer International Publishing, Cham, 415–428. https://doi.org/10.1007/978-3-319-11197-1_32

[18] Avishay Traeger, Erez Zadok, Nikolai Joukov, and Charles P. Wright. 2008. A Nine Year Study of File System and Storage Benchmarking. *ACM Transactions on Storage* 4, 2 (May 2008), 1–56. https://doi.org/10.1145/1367829.1367831

[19] Theodore Ts'o. [n. d.]. *Event Tracing — The Linux Kernel documentation.* https://docs.kernel.org/trace/events.html Accessed: 2023-02-27.

[20] Simon Volpert. 2024. Workflows: An Empirical Analysis of Common OCI Runtimes' Performance Isolation Capabilities. https://doi.org/10.5281/zenodo.10612048

[21] Simon Volpert, Benjamin Erb, Georg Eisenhart, Daniel Seybold, Stefan Wesner, and Jörg Domaschka. 2023. A Methodology and Framework to Determine the Isolation Capabilities of Virtualisation Technologies. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*. ACM, Coimbra Portugal, 149–160. https://doi.org/10.1145/3578244.3583728

[22] Zhiyuan Wan, David Lo, Xin Xia, and Liang Cai. 2019. Practical and Effective Sandboxing for Linux Containers. (2019), 41. https://doi.org/10.1007/s10664-019-09737-2

[23] Kun Wang, Song Wu, Kun Suo, Yijie Liu, Hang Huang, Zhuo Huang, and Hai Jin. 2023. Characterizing and Optimizing Kernel Resource Isolation for Containers. *Future Generation Computer Systems* 141 (April 2023), 218–229. https://doi.org/10.1016/j.future.2022.11.018

[24] Xingyu Wang, Junzhao Du, and Hui Liu. 2022. Performance and Isolation Analysis of RunC, gVisor and Kata Containers Runtimes. *Cluster Computing* (Jan. 2022). https://doi.org/10.1007/s10586-021-03517-8

[25] Miguel G. Xavier, Israel C. De Oliveira, Fabio D. Rossi, Robson D. Dos Passos, Kassiano J. Matteussi, and Cesar A.F. De Rose. 2015. A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. 253–260. https://doi.org/10.1109/PDP.2015.67

[26] Ethan G Young, Pengfei Zhu, Tyler Caraza-Harter, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. 2019. The True Cost of Containing: A gVisor Case Study. (2019), 6.