Exemplary Determination of Cgroups-Based QoS Isolation for a Database Workload

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

Daniel Seybold
Jörg Domaschka
daniel.seybold@benchant.com
joerg.domaschka@benchant.com
BenchANT GmbH
Ulm, Germany

ABSTRACT

An effective isolation among workloads within a shared and possibly contended compute environment is a crucial aspect for industry and academia alike to ensure optimal performance and resource utilization. Modern ecosystems offer a wide range of approaches and solutions to ensure isolation for a multitude of different compute resources. Past experiments have verified the effectiveness of this resource isolation with micro benchmarks. The effectiveness of Quality of Service (QoS) isolation for intricate workloads beyond micro benchmarks however, remains an open question.

This paper addresses this gap by introducing a specific example involving a database workload isolated using Cgroups from a disruptor contending for CPU resources. Despite the even distribution of CPU isolation limits among the workloads, our findings reveal a significant impact of the disruptor on the QoS of the database workload. To illustrate this, we present a methodology for quantifying this isolation, accompanied by an implementation incorporating essential instrumentation through Extended Berkeley Packet Filter (eBPF).

This not only highlights the practical challenges in achieving robust QoS isolation but also emphasizes the need for additional instrumentation and realistic scenarios to comprehensively evaluate and address these challenges.

CCS CONCEPTS

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

KEYWORDS

Isolation, Performance, eBPF, Benchmarking, Cloud, DBMS

ICPE '24 Companion, May 7–11, 2024, London, United Kingdom

ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652267

ACM Reference Format:
Simon Volpert, Sascha Winkelhofer, Stefan Wesner, Daniel Seybold, and Jörg Domaschka. 2024. Exemplary Determination of Cgroups-Based QoS Isolation for a Database Workload. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3652267

1 INTRODUCTION

In the ever-evolving landscape of computing, the paradigm shift toward cloud computing and larger-scaled compute environments has revolutionized the way organizations deploy, manage, and utilize computing resources. Cloud computing, in particular, offers unparalleled scalability, flexibility, and cost-effectiveness, enabling businesses and scientists to work on challenges that were unattainable without it [14, 17]. However, the shared nature of resources in such environments introduces inherent challenges, necessitating robust mechanisms to ensure isolation among disparate workloads and tenants. These challenges can be imposed by the desire to consolidate physical hardware, overbooking or overcommitting as a business model, or by misbehaving disruptive tenants acting as "noisy neighbors".

There is a wide range of solutions that aim to solve these isolation challenges. One aspect towards a solution is various virtualization technologies. These range from classic hypervisor-based implementations over manifold container-based solutions towards more recent developments in the concept of application sandboxing. Many of them pursue different strategies to achieve adequate isolation; however, they do share some commonalities. A frequently used strategy is the utilization of Cgroups [7, 23].

Cgroups are provided by the Linux kernel. They enable an operator to distribute processes into groups and subsequently assign resource limits to those groups. These mechanisms have proven to work very well, specifically when solely observing the isolated and limited resource. The patterns of resource usage of real-world applications are often more complex [5]. Their QoS is not necessarily directly dependent on a few distinct resources, as it is a measure of end-to-end performance that inherently involves any amount of resources [4, 16].

In this paper, we focus on Cgroup-based CPU isolation. For this specific case, we investigate whether one tenant's QoS is impacted by another disrupting tenant, even though their CPU limits are evenly shared with no overbooking in place. With this, we aim at answering the following two research questions:

RQ 1 (Isolation measurement). *How can QoS isolation be challenged in complex deterministic scenarios?*

RQ 2 (Cgroup sufficiency). *Is Cgroup-based isolation enough for reliant QoS isolation?*

Answering these questions, we provide several contributions. First, we provide a strategy to measure isolation between two tenants considering the impact on QoS. Second, we suggest metrics that quantify the degree of isolation. Finally, we provide a tool developed for this work that enables low-overhead instrumentation of compute resources for isolated process trees.

The remainder of this paper is structured as follows. In section 2 we discuss the fundamentals of this work. This includes eBPF profiling, Cgroups and a discussion of isolation and its quantification for QoS. This is followed by a description of the methodology applied in section 3 and lays the foundation for the answer to **RQ 1**. The methodology is followed by important details of the implementation in section 4. It gives a brief overview of the technologies involved in the experimental setup and the workflow of measured scenarios. The final results in section 5 discuss the observations and in this process answers **RQ 2**. We close with a review of related work in section 6 and a final summary in section 7.

2 BACKGROUND

This section describes important background aspects for the subsequent progression of this work. This includes low-overhead instrumentation, Cgroups, and isolation considerations.

2.1 Linux Profiling with eBPF

The Linux profiling subsystem efficiently gathers and collects performance data, enabling developers and operators to pinpoint and enhance resource utilization patterns. Retrieving these comes with a performance penalty depending on the method of accessing it.

eBPF facilitates the execution of verified code within a dedicated Virtual Machine (VM) integrated in the Linux kernel, extending the capabilities of the original Berkeley Packet Filter (BPF) [13]. Beyond executing functions upon receiving network packets, eBPF can observe and respond to various event sources as part of the Linux profiling subsystem, including Performance Monitoring Counters (PMCs), tracepoints, and both kernel and user functions.

Although these events are not technically part of eBPF, it provides an accessible means of leveraging them. Specifically, the instrumentation and processing of profiling data directly within the kernel space can reduce instrumentation overhead, since frequent interactions with kernel and userspace are kept to a minimum.

The typical lifecycle of an eBPF program is depicted in fig. 1, as presented by Gregg [6]. As depicted here, a typical first step is the *(i)* generation of BPF byte-code by arbitrary eBPF tooling. Upon this generation, the byte-code is *(ii)* loaded into the kernel for a verifying step before being passed to the eBPF VM. For exchanging data between Kernel- and userspace, the *(iii)* perf_output and *(iii)* async read channels can be utilized.

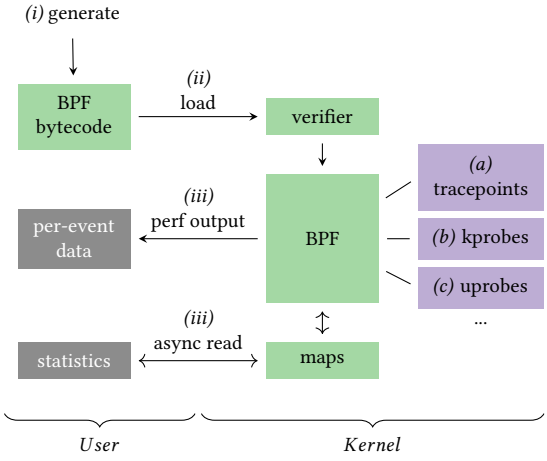

Figure 1: eBPF internals and Linux instrumentation according to [6]

Within the scope of this work, we are employing instrumentation on *(a)* Tracepoints. Tracepoints are static points of kernel instrumentation [19], established and implemented by kernel developers to trigger an event upon a specific call. They also incorporate hardware-specific counters, such as CPU cycles per core since boot time.

eBPF based instrumentation is naturally tightly coupled with the currently loaded Linux kernel. The BPF Type Format (BTF) aims to improve the portability of eBPF based tools by providing a metadata format, which encodes debug information related to the functions and structures of the kernel referenced in the eBPF programs. The profiling tool trac[1], developed during this work, utilizes this format. The tool itself is in an ongoing development phase.

This section only briefly outlines eBPF and Linux profiling, with a more detailed exposition available in the previous work of fellow authors [2, 20].

2.2 Cgroups

Control groups[2] are a Linux feature that enables precise control over the utilization of various system resources [8]. The Linux kernel ensures that the processes assigned to such a group adhere to the limits specified for the Cgroup. Additionally, Cgroups can be unique, shared, and nested, essentially creating a hierarchical structure.

Cgroups offer powerful measures to control, limit, and possibly isolate resources. Used in conjunction with namespaces, they act as an essential enabler for virtualization, particularly in the context of container virtualization [20].

The Cgroups project underwent a significant restructuring effort, resulting in the recent release of Cgroups v2. This effort was first merged into the kernel with version 4.5 and is able to fully replace v1

[1] https://github.com/omi-uulm/trac
[2] https://man7.org/linux/man-pages/man7/cgroups.7.html

since kernel version 5.6 [3]. At the time of writing, the list of Cgroup controllers include *(i)* cpu, *(ii)* cpuset, *(iii)* freezer, *(iv)* hugetlb, *(v)* io, *(vi)* memory, *(vii)* perf, *(viii)* pids, and *(ix)* rdma.

This work focuses on the usage of the *(i)* CPU controller implemented by Cgroups v2. It enables setting a limit on the number of CPU cycles per second.

2.3 Isolation Terminology

Isolation is a condition that occurs when two workloads share a resource and compete for it. The degree to which they interfere with each other characterizes isolation. If their influence on each other is distinctive, the isolation is considered low, and vice versa. This concept is discussed by several authors [10, 12, 22]. This study follows the definition of isolation provided by Krebs et al. who define:

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, particularly in cloud computing, the term "noisy neighbor" is often used in related literature. This term refers to a disruptive tenant that adversely affects another tenant. According to the definition provided by Longbottom [11], a noisy neighbor is described 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 QoS Isolation Quantification

Performance degradation is a measure of how strong an abiding workload W_a is affected by a disruptive workload W_d. It can be determined as "performance loss rate" I_{plr} [9, 12, 18, 22].

$$I_{plr} = \frac{|W_{a_1} - W_{a_2}|}{W_{a_1}} \tag{1}$$

Here W_{a_1} represents a workload in an undisrupted environment, whereas W_{a_2} represents the same workload impacted by a disruptive workload W_d.

Krebs et al. extends this simple model with one specifically targeted at QoS isolation determination [10]. We apply and slightly adapt this model to fit our measured parameters.

Taking eq. (1) as a basis, we can determine the actual performance ratio q_{W_a} and q_{W_b} by calculating $1-I_{plr}$. This leads to the simplified eq. (2) and eq. (3).

$$q_{W_a} = \frac{W_a}{W_{a_{ref}}} \tag{2}$$

$$q_{W_d} = \frac{W_d}{W_{d_{ref}}} \tag{3}$$

Using eq. (2) and eq. (3) we can then determine the remaining relative performance ρ at a certain q_{W_d} as q_{W_a}.

$$\rho(q_{W_d}) = q_{W_a} \tag{4}$$

Table 1: Scenarios

name	W_a	l_a	W_d	l_d
(i) baseline	100 %	50 %	0 %	50 %
(ii) harmony	100 %	50 %	0-100 %	50 %

As Krebs et al. further states, these kind of values represent only a distinct point where the disruption is to a specific degree. To address this, we can try to reduce the resulting series of eq. (4) to a single isolation metric I.

An approach is to limit the number of samples q_{W_d} to m equidistant points and subsequently compute their arithmetic mean:

$$I_{avg} = \frac{\sum \rho(q_{W_d})}{m} \tag{5}$$

As this likely neglects the maximum amount of degradation, we can further derive another metric that describes the maximum isolation impact I_{max} as follows:

$$I_{max} = \frac{\min(q_{W_a})}{\arg\min\limits_{q_{W_d}}(q_{W_a})} \tag{6}$$

Naturally, employing either eq. (5) or eq. (6) might overlook the inherent curve of I_{QoS}, potentially introducing a bias to the outcome. Further considerations on deriving a metric that avoids this are left for future work.

3 METHOD

This section presents the method behind the conducted experiments and thus elaborates on the scenarios, instrumentation, and isolation quantification. These aspects are adapted from previous work [20, 21].

Goal. As mentioned in section 1 we aim to measure the Cgroup QoS isolation for the CPU resource. According to section 2.4 we need at least two distinct measurements to analyze the isolation capability of a technology. One being the reference workload in an uncontended environment, and the other being the same workload under contention.

Scenarios. As we are interested in whether a QoS-based isolation is as high as a specific isolation for a certain resource, we choose an appropriate isolation scenario. Earlier work has shown that this is the case for fairly distributed resources where no overbooking, overcommitting, or aggressive resource stealing happens [20, 21]. Volpert et al. show that this is particularly true for the "harmony" scenario.

Therefore, this work analyzes the isolation of two scenarios: *(i)* baseline and *(ii)* harmony. These are itemized in table 1

Here, W_a and W_d describe the workload performed within their respective imposed limits l_a and l_d. W_a is considered static in both scenarios and is instrumented regarding its consumed resources and QoS status. It is further supposed to resemble a realistic workload and is thus realized as a macro or synthetic benchmark [9]. For the *(ii)* harmony scenario, W_d gradually increases over time and is also instrumented for its consumed resources. Its purpose is to specifically stress the single resource that is being isolated.

Instrumentation. Again, the resource instrumentation approach follows the principles outlined in previous work by the authors [20, 21]. In summary, it is independent of isolation technology and performed outside of the isolation group. This is achieved with eBPF.

Isolation quantification. In section 2.4, we introduce and briefly examine metrics for quantifying QoS isolation. Utilizing eBPF and QoS metrics reported by W_a we can quantify the isolation at specific degrees of contention by W_d.

4 EXPERIMENT DESIGN

In this section, we describe the abstract workflows of the experiments. These are followed by a presentation and reasoning behind the choices for the tools and instrumentation points selected.

4.1 Experiment workflow

As described in section 3, the experimental workflow follows two scenarios. The execution of a scenario is highlighted in fig. 2

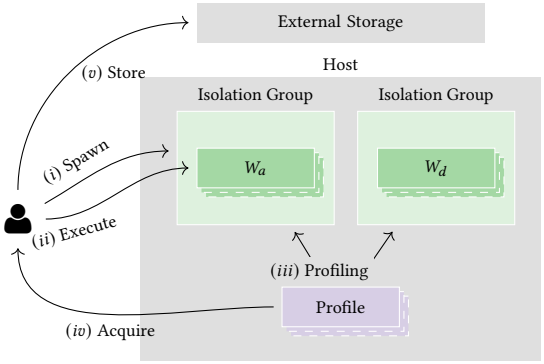

Figure 2: Flow of an abstract measurement

The process begins with *(i)* the initialization of an isolation group. In this phase, *(ii)* load is generated by W_a and W_b. The *(iii)* profiling process on the host system is initiated concurrently. This profiling monitors the isolation groups. Upon completion, data is *(iv)* collected and *(v)* stored on external storage.

Each run takes 5 minutes and is repeated 3 times. After each run, the whole physical systems are reset and pruned to guarantee no unintended side effects by residue of past experiments and improving reproducibility.

4.2 Approach and Implementation

The following briefly iterates over the actual implementation of the method as described in section 3 is realized.

Load generation. As stated in section 3 we need two distinct workloads W_a and W_d. W_a is supposed to act as a realistic workload. Here, we choose to run a YCSB[3] benchmark on a remote host against a Postgres database [1]. The throughput in operations per second and thus the QoS workload W_a is determined for this databse. For the sake of simplicity, we choose an insert-only workload stressing the database for 5 minutes. In that 5 minutes, YCSB tries to execute 100, 000, 000 inserts of 500 bytes with 90 threads. After its run-time, it reports a list of all operations with timestamp and latency. Operations per second can be derived by resampling to a desired frequency and counting the operations. These operations per second are considered to be the QoS metric of W_a.

The Postgres database is continually instrumented with respect to its CPU cycles and operations per second. Its configuration is generated by PGtune[4] optimizing Postgres with half of the total resources available on the physical server as described in section 5.1[15].

W_d is considered to be a micro benchmark that continuously stresses the CPU. We use the `stress-ng` implementation to realize that load. It is set up such that it increases its utilization over time, until it fully utilizes its granted resources. To achieve a linear load generation behavior, we partition this load generation into multiple intervals with configurable resolution.

Assuming ideal isolation, the measures of both workloads resemble a progression, as illustrated in 3.

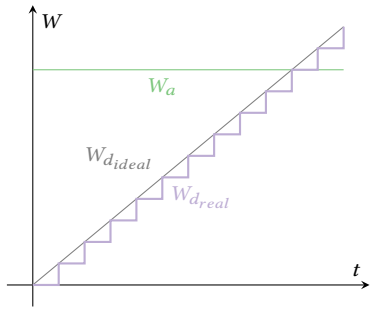

Figure 3: Load generation

Instrumentation. Since we focus on CPU isolation, we select an instrumentation point as outlined in section 2.1 that gives a detailed view on CPU utilization. Modern CPUs provide hardware-based counters that report the cycles that are executed on each core. The progression of the counters over time, along with the maximum number of possible cycles per core, can be used to derive CPU utilization.

In order to keep the instrumentation overhead as low as possible, we opt to utilize eBPF instrumentation. This allows us to gain fine-grained control over the sampling frequency and efficient profiling inside the kernel space. To leverage eBPF instrumentation, we built a profiling tool named "trac[5]".

Trac allows to be attached onto a root process. This root process and any process invoked by it are subsequently instrumented for either CPU cycles, resident memory, disk I/O, or network I/O. The gathering of those metrics happens inside the kernel space, where they are collected in a datastructure, called a BPF map. After profiling, these maps can be accessed by the user-space counterpart of the profiling tool. The collected data are processed and presented as CSV time series with a resolution of up to $1ms$.

[3] https://github.com/brianfrankcooper/YCSB

[4] https://pgtune.leopard.in.ua/

[5] https://github.com/omi-uulm/trac

Isolation. To isolate processes with Cgroups, we leverage the isolation tool "nsJail[6]". NsJail is a Linux process isolation tool that utilizes the Linux namespace subsystem, Cgroup resource limits, and seccomp-bpf syscall filters to achieve process isolation.

In particular, we use the tool's Cgroup capabilities to isolate the stress-ng CPU load generator, as well as the Postgres database.

The stress-ng CPU load generator itself does not utilize other system resources such as memory, disk I/O and network I/O. As a consequence, we do not isolate these between workloads. Moreover, memory, disk I/O, and network I/O utilized by the Postgres database are negligible for the configuration and workload applied.

5 EVALUATION

This section systematically discusses the results of the evaluation outlined in the sections before.

5.1 Evaluation Environment

The experimental configuration encompasses a pair of physical servers, symmetrically arranged and equipped with identical components. Both servers feature two Intel CPUs, specifically the "Intel(R) Xeon(R) CPU E5-2630 v3", operating at a base clock frequency of 2.40 GHz with 32 cores. Memory associated with these CPUs totals $16 \cdot 16 = 256$ GiB of DDR4 memory clocked at 2133 MHz. The physical storage disk for the database state is a Samsung SM843TN, which exhibits a Input Output Operations Per Second (IOPS) performance of 15, 000 for "random write" operations.

Communication for actual workload between all nodes is separated and facilitated by Mellanox Technologies' Network Interface Card (NIC) from the "MT27800 ConnectX-5" family, capable of a network throughput of 50 Gbit/s.

Figure 4 visualizes the interaction between the pair of physical servers mentioned above. Here YCSB is responsible to generate and control W_a (Postgres) from a remote host. We do so to limit possible interference on W_a by the load generated by YCSB. The latter should not be accounted for as it would act as a "noisy neighbor". W_d generates its own workload and is not externally controlled.

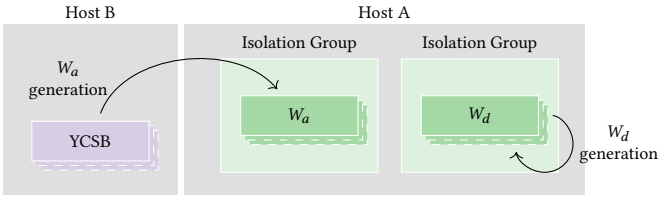

Figure 4: Workload generation and controlling across hosts

The complete experiment set-up includes additional auxiliary servers responsible for workflow automation. Notable involved software components are itemized in table 2.

5.2 Results

In the following, we iteratively discuss the results of the scenarios presented in table 1. Each scenario is represented by a plot.

[6]https://github.com/google/nsjail

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	v3.5.4	The workflow engine to orchestrate experiments and scenarios
trac	0.2.3	Profiling tool based on eBPF and Aya
stress-ng	0.13.05	Load generator for CPU
YCSB	0.17.0	Macro benchmark for databaes
Postgres	15	SQL based Database management engine
nsjail	3.4	Process isolation utilizing Linux kernel functions

Table 2: Software version list

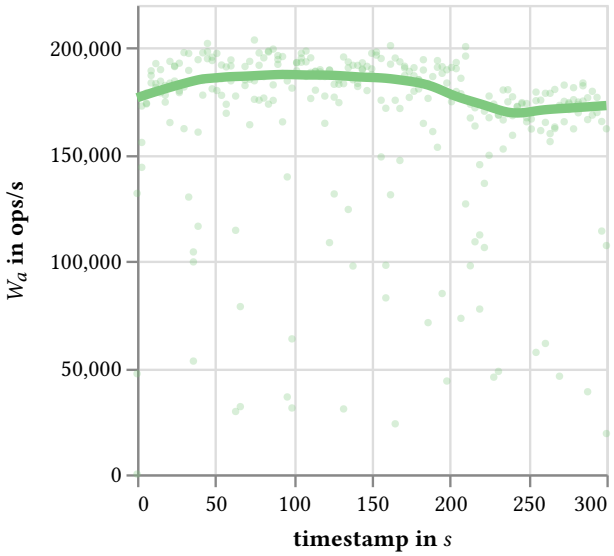

Figure 5: Baseline scenario

For the actual isolation metric determination we present an additional graph highlighting the impact on W_a QoS isolation at every observed degree of stress imposed by the disruptive workload W_d.

Figure 5 visualizes the baseline scenario. The y-axis shows W_a in operations per second at a given interval in seconds. As described above, this information is provided by YCSB. As each experiment is repeated multiple times and actual operations per second are volatile, we adapted the visualization accordingly. Every measured data point is plotted as a small circle resulting in a scatter plot. An overlay as a smoothed thicker line highlights the trend of those data points. The smoothing algorithm applied implements the Locally Estimated Scatterplot Smoothing (LOESS) method. This results in a trend for this baseline graph that settles roughly at $175,000 ops/s$.

The visualization method for W_a in fig. 6 follows the same principle. However, the visualization of the disruptive workload W_d does not apply said algorithm. Instead, it plots an overlays as the mean

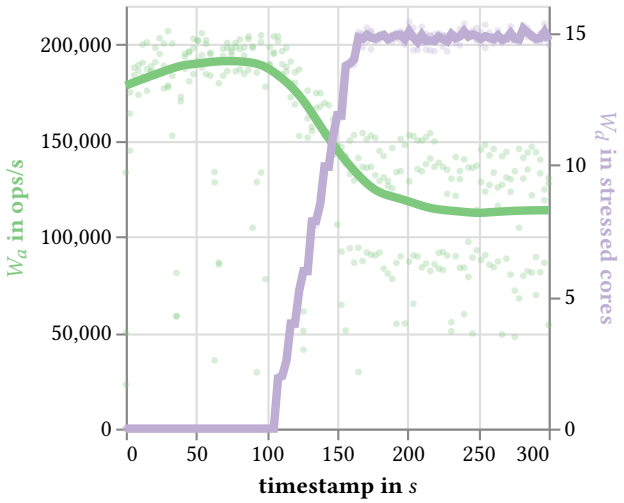

Figure 6: Harmony scenario

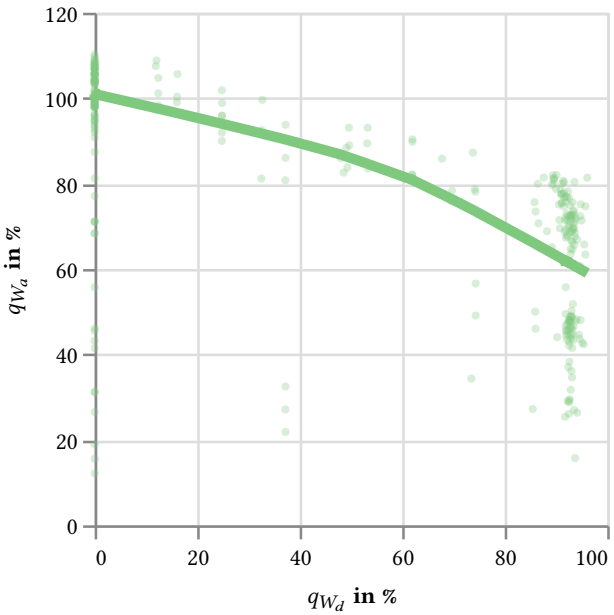

Figure 7: Isolation impact

of repeated runs, as these measurements are stable. Thus, the steps of the gradually increasing disruptive workload are easily visible.

Adding this disruptive workload W_d has a significant impact on the behavior of W_a. After an initial pausing duration of 100s we can see an immediate degradation of ops/s for W_a. This gets worse as W_d reaches its full utilization and results in a degradation of W_a of almost 50%.

Most importantly, neither workload ever exceeds 50% of the physical system capacity, as defined by its assigned CPU cycle limit. This means that the CPU cycles isolation works well considering the fact that no workload is able to exceed its limit. This is in direct conflict of the 50% QoS degradation observed. It is evident that a harmonic split of the seemingly available total resource of CPU cycles can have an impact on each other's CPU performance.

A more detailed visualization with a specific focus on the impact on isolation is presented in fig. 7. Here, the x- and y-axes represent the relative degradation ratio of the workload as defined in section 2.4 with the dimension of time completely removed. Therefore, this graph represents q_{W_a} for every q_{W_d}. Again, because of the volatile nature of the measure points, we present the graph as a trend overlay over a scatter plot. Here, we can see a slight change in the degree of degradation above 50% of q_{W_d}. What is also easily visible here is that good isolation between W_a and W_d is represented by a higher value, while worse isolation is represented by a lower value within the interval of $[0, 1]$. In table 3 discrete interesting values of fig. 7 are presented. Taking into account the equidistant q_{W_d} values in the interval $[0.1, 0.9]$ of this table results in $\mathbf{I_{avg} = 0.83}$ for eq. (5). Furthermore, we can also calculate $\mathbf{I_{max} = 0.60}$. These values are naturally different from each other, as they both describe different properties of the isolation function ρ.

The observations above lead to the following interpretation.

Interpretation. The reason behind the observation that W_d can have such a huge impact on W_a even though they should not impact each other can be manifold. However, two aspects seem to play an important role here.

Table 3: Isolation metrics comparison

q_{W_d} in %	q_{W_a} in %
0.0	93.6
10.0	92.0
20.0	89.7
30.0	87.5
40.0	86.0
50.0	84.7
60.0	82.6
70.0	78.8
80.0	72.2
90.0	62.5
95.0	56.8

The system we execute our experiment on features two hyperthreading enabled CPUs. Theoretically speaking, they are able to fully utilize all logical cores with maximum cycles if the workload fits. This was observed in previous work of the authors [20]. However, the workload in terms of QoS decreases significantly when the actual physical cores are fully utilized. This assumption is indicated by the slight change of slope in fig. 7 at $q_{W_d} \approx 50\%$. Although more cycles could be utilized by the respective workloads while staying within their cycle limit, they are not able to use them to maintain their QoS.

Another limiting factor could be due to the saturation of only loosely related resources in regard to CPU cycles. This could be due to the overhead induced by process scheduling. Thus, CPU cycles

could be increasingly reserved for such essential tasks, leading to even more starvation of W_a.

6 RELATED WORK

A prevalent method for assessing the isolation capability involves calculating the I_{plr} as outlined in eq. (1). In line with this approach, previous studies commonly determine this on a per-resource basis [12, 18, 22, 23]. We extend these findings with considerations regarding QoS.

Silva et al. reviewed the effectiveness of resource isolation for QoS isolation in the past [16]. They state that providing QoS for application performance requires more than just guaranteeing a certain allocation of CPU, memory, or I/O resources. We support their findings for the more recent Cgroups v2 and extend them with further measurements and an isolation quantification model.

7 CONCLUSION

Over the course of this work, we designed and implemented a sophisticated experimental setup that allowed us to execute two workloads against each other in order to measure their isolation from each other. We have deliberately chosen a very specific scenario, where a synthetic "abiding" database under constant workload competes against a "disruptive" stressor that utilizes the CPU as high as possible.

We determine that those two workloads influence each other even when their CPU limits are evenly shared across the available resources without any overbooking. Neither workload exceeds its limit, but the impact on the QoS of the abiding database is clearly visible.

As a consequence, we can see that mere CPU isolation is insufficient for more complex workloads outside of micro-benchmarks that try to escape them. Aspects like hyper-threading and CPU scheduling overhead are CPU related resources that are not isolated as probably expected. Applying these findings to real-world scenarios requires in situ system tests to determine the actual impact on QoS when co-locating tenants.

The results presented in this work can be considered as a first preliminary step towards more effective QoS isolation. From this point on we see various possible future directions. One is the configuration of stricter isolation environments with limited hyper-threading and possibly system call filtering mechanisms of sandboxes. Another direction could be the improvement of instrumentation to pinpoint the actual saturated resource resulting in a drop in QoS. Lastly, those considerations could be repeated for other Cgroup, different isolation technologies or other workloads.

REFERENCES

[1] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*. ACM, Indianapolis Indiana USA, 143–154. https://doi.org/10.1145/1807128.1807152

[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] Chris Down. 2021. 5 Years of Cgroup v2: The Future of Linux Resource Control. USENIX Association.

[4] CSRC Content Editor. [n. d.]. Quality of Service (QoS) - Glossary | CSRC. https://csrc.nist.gov/glossary/term/quality_of_service.

[5] Siqian Gong, Beibei Yin, Zheng Zheng, and Kai-Yuan Cai. 2019. Adaptive Multivariable Control for Multiple Resource Allocation of Service-Based Systems in Cloud Computing. *IEEE Access* 7 (2019), 13817–13831. https://doi.org/10.1109/ACCESS.2019.2894188

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

[7] Leila Helali and Mohamed Nazih Omri. 2021. A Survey of Data Center Consolidation in Cloud Computing Systems. *Computer Science Review* 39 (Feb. 2021), 100366. https://doi.org/10.1016/j.cosrev.2021.100366

[8] Tejun Heo, J Weiner, V Davydov, L Thorvalds, P Parav, T Klauser, S Hallyn, and K Khlebnikov. 2015. Control Group V2. https://www.kernel.org/doc/Documentation/admin-guide/cgroup-v2.rst.

[9] 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

[10] 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

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

[12] 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

[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] Cristian Ruiz, Emmanuel Jeanvoine, and Lucas Nussbaum. 2015. Performance Evaluation of Containers for HPC. In *Euro-Par 2015: Parallel Processing Workshops (Lecture Notes in Computer Science)*, Sascha Hunold, Alexandru Costan, Domingo Giménez, Alexandru Iosup, Laura Ricci, María Engracia Gómez Requena, Vittorio Scarano, Ana Lucia Varbanescu, Stephen L. Scott, Stefan Lankes, Josef Weidendorfer, and Michael Alexander (Eds.). Springer International Publishing, Cham, 813–824. https://doi.org/10.1007/978-3-319-27308-2_65

[15] Daniel Seybold and Jörg Domaschka. [n. d.]. PostgreSQL - Configuration Tuning. https://benchant.com/en/blog/postgresql-configuration-tuning.

[16] Marcio Silva, Kyung Dong Ryu, and Dilma Da Silva. 2012. VM Performance Isolation to Support QoS in Cloud. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE, Shanghai, China, 1144–1151. https://doi.org/10.1109/IPDPSW.2012.140

[17] Ali Sunyaev and Ali Sunyaev. 2020. Cloud computing. *Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies* (2020), 195–236.

[18] 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

[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, 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

[21] Simon Volpert, Sascha Winkelhofer, Stefan Wesner, and Jörg Domaschka. 2024. An Empirical Analysis of Common OCI Runtimes' Performance Isolation Capabilities. In *Proceedings of the 2024 ACM/SPEC International Conference on Performance Engineering*. ACM, London United Kingdom. https://doi.org/10.1145/3629526.3645044

[22] 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

[23] 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