Performance losses with virtualization: Comparing bare metal to VMs and containers

Jonatan Baumgartner[0009-0003-1372-9693], Christophe Lillo and Sébastien Rumley[0000-0001-5547-9483]

School of Engineering and Architecture of Fribourg,
HES-SO - University of Applied Sciences and Arts Western Switzerland
sebastien.rumley@hefr.ch

Abstract. The use of virtualization technologies has become widespread with the advent of cloud computing. The purpose of this study is to quantify the performance losses caused by all kind of virtualization/containerization configurations.

A benchmark suite consisting of tools that stress specific components and then four real applications commonly used in computing centers has been designed. A system to schedule the execution of these benchmarks and to collect the results has been developed. Finally, a procedure calling all the benchmark in a consistent and reproducible way either within a container or in a (virtual or not machine) has been implemented. These developments permitted then to compare bare metal with four hypervisors and two container runtimes as well as the mix of containers in the virtual machines.

The results show that the performance differences vary greatly depending on the workload and the virtualization software used. When using the right virtualization software, the estimated the performance losses are around 5% for a container and 10% for a virtual machine. The combination of the two entails the addition of these losses to 15%. In the case of non-optimized software, a performance loss of up to 72% can be observed. We also observed that containers and virtual machines can over-perform bare-metal when it comes to file access.

Overall we conclude that virtualization has become very mature and performance losses seems not to be a concern anymore.

Keywords: virtualization, benchmarking, overhead, measures, automation.

1 Introduction

During the last decade, containerized or virtualized applications have gradually replaced bare metal computers. These techniques have been popularized by the groundbreaking arrival of cloud computing in companies and public institutions. Hence it is now common for a company to rely exclusively on virtual machines (in addition to personal computers), hosted in a private cloud, or rented to a cloud company. Companies and institutions also increasingly resort to higher-level virtualization solutions, be it thru Containers as a Service (Caas) or Functions as a Service (FaaS). They can even

turn to Kubernetes as a Service (KaaS) solutions to manage container deployment themselves, but without having to manage any of the inferior levels (hardware, OS).

Stacking services has become so easy that it is common to find multiple layers of virtualization coexisting. For instance, Switch, a swiss academic IT provider, offers a KaaS solution that itself rely on virtual machines (Switch Engine) [16]. It has also become a frequent practice to perform docker-in-docker operations or install docker inside virtual machines for convenience. Furthermore, there is a growing need for nested virtualization [17], which involves running virtual machines inside another.

Virtualization - of the hardware machine (VM hereafter) or of the OS (container hereafter) - makes a lot of sense when it comes to increasing utilization, guaranteeing reproducibility, or easing the deployment. However, there is no free lunch and these benefits come at a cost, namely in terms of performance overhead. We therefore posit that it is legitimate to ask how important this overhead is.

Moreover, in light of the recent energy crisis and the broader objective of transitioning to carbon-neutral societies, the aspect of electrical consumption of IT is gaining importance. Consequently, we inquire: in an energy sobriety context, can a debauchery of virtualization significantly affect the energy consumption of our computing resources? Stated in another way: how much longer will my virtualized application keep my CPU busy? And in yet other terms: what is the performance overhead introduced by virtualization, compared to running the same application directly on the hardware? Addressing the latter question forms the focus of our research, which we explore and discuss in this paper.

Much research in the past tried to answer this question. One of the main challenges for virtualization is the storage input and output rate [1] but other comparisons can be made between bare-metal and high- or low-level virtualization [2]. Some papers [3, 4, 5] are quite old and use obsolete technologies like OpenVZ, LXC or Linux-Vserver. Other studies [6, 7] only concentrate on VMs or containers.

The applications used to benchmark the performances differ. While HPL, STREAM and iPerf are almost always used to test the CPU [1, 6, 9], the memory and the network, there is not one default choice when it comes to measuring storage performances. Several articles focus on a specific real-world application: compiling a Linux kernel, testing the capabilities of a web server with RUBiS [3], or accesssing Cassandra database [10]. Shirinbab et al. [11] found that when comparing VMware to XenServer and KVM, no hypervisor has the best performance for all aspects.

Table 1 compiles the results found in literature [2, 3, 4, 9]. We see that if there is a consensus on performance degradation for CPU demanding application, there is much

Table 1. Summarized performances loss found by previous studies.

	BARE METAL	CONTAINER	VM
CPU	100%	90-100%	90-100%
MEMORY	100%	95-100%	69-100%
DISK IO	100%	50-90%	45-100%
NETOWRK	100%	64-100%	47-100%

more debate about memory or disk intensive workloads. Some papers report discouraging performances in the range of 50% (meaning virtualization introduces a 50% overhead), while others conclude that it is not so bad (at least 90% of the performance).

Interestingly, we note that if there has been a volley of papers published on virtualization benchmarking in the years 2007-2015, the subject has somehow lost traction in the recent years. In our opinion, this is unfortunate since the virtualization technologies did evolve in the last years, notably on the storage performance side, for example with the apparition of ZFS and its aggressive caching methods in Proxmox [12] or with the implementation of virtiofs in Docker [13]. Another type of improvements is the apparition of lighter Kubernetes distributions like k0s which leave more resources available for the containerized workloads themselves. It is also interesting to check if the most recent platforms still hardly affect performances in peculiar cases, as the one reported by Morabito et al. [9] (UDP traffic).

In summary, we made the decision to carry out a new campaign of benchmarks, with the goal of reconciling the diverse range of results found in the literature and definitively addressing our research question. Our campaign consisted of comparing various virtualization platforms and combination thereof on three different hardware, running a suite of benchmarks. Altogether we totalized more than 1'000 hours of benchmarking, yielding in more than 15'000 measurements. To automate the conduction of the experiment, we have implemented a comprehensive test orchestration framework called LSBS, which we made open source available.

Our benchmarking methodology and test orchestration framework LSBS that implements it is presented in Section 2. Section 3 presents the results obtained when conducting the tests on the three different hardware targets. We discuss the collected results in section 4. Section 5 concludes.

2 Methodology

We divided our methodology in three components: the *benchmark suite* itself, consisting of different workloads whose performance are measured; the *benchmark procedure*, responsible for (repetitively) calling the *suite* on every *target* available in every desired virtualization configuration, and finally, a data collection system in which all measurements are centralized.

2.1 Benchmark suite

We strived to assemble a comprehensive and representative collection of benchmarks. The components of this suite are outlined in Table 2. The suite begins with benchmarks that assess individual "hardware" components (IDs 1-16). The HPCC benchmark suite (IDs 1-4) evaluates the CPU and RAM through the utilization of HPL, DGEMM, RandomAccess, and Stream tests. FIO (IDs 5-12) has been chosen for storage testing, encompassing eight measurements that include random or sequential operations, reads or writes, and recording either the operations per second or the speed. Lastly, for network (IDs 13-16), ping is used to measure latency, and IPerf is employed for network assessment, initially in normal mode, followed by reverse mode, and ultimately in UDP mode.

Table 2. The 20 components of the developed benchmark suite

ID	COMPONENT	BENCHMARK	MEASUREMENT	UNIT
1	CPU	HPL	Rate of operations performed by the CPU while resolving a large double precision linear equation system	FLOPS
2		DGEMM	Rate of floating-point operations performed by the CPU multiplicating large matrixes	FLOPS
3	Memory	RandomAccess	Rate of random integer update in the memory	Up/s
4		Stream	Sustainable memory bandwith	GB/s
5	Storage	FIO	Sequential write iops	Io/s
6			Random write iops	
7			Sequential read iops	
8			Random reand iops	
9			Sequential write speed	Kb/s
10			Random write speed	
11			Sequential read speed	
12			Random reand speed	
13	Network	Ping	time between tested and control system	ms
14		Iperf	Network TCP speed from target to control system	Gb/s
15			Network TCP speed from control to target system	
16			Network UDP speed from target to control system	
17	Compound	Blender	Time used to perform the task	seconds
18		Database		
19		Deep Learning		
20		REST server		

In the second part of the suite (IDs 17-20), 4 real applications are used: the rendering of a Blender scene with Blender 3.3.0 and the Classroom scene; operations on a 48MB SQLite database containing phone calls with their source, destination, and cost; the training of a generative adversarial neural network with 322 images; a REST server which, upon requests, creates a 70 MB random file and then asks the client to download it. We measure the execution times of these test applications.

Running the full suite requires multiple launch commands. Installing the suite also requires multiple operations. To both simplify, expedite and, very importantly, standardize both the installation and conduction processes, we described them as Ansible playbooks. Ansible [14] is a command-line software purposed for software installation automation. These playbooks (install and run) can then be "played" over a freshly installed Ubuntu OS. This methodology applies for bare-metal and VM benchmarking.

For containers benchmarking, we wrote Dockerfiles and built images for each benchmark, as well as Ansible playbooks that installs the container runtime and runs the benchmark suite.

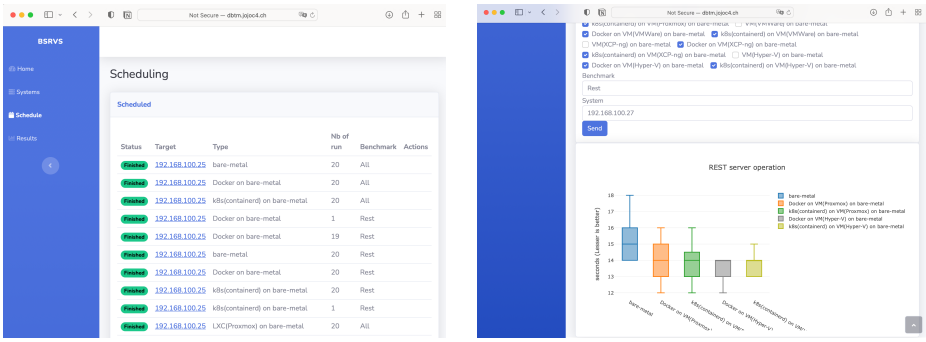
2.2 Benchmarking procedure

In order to run the benchmark suite not only on multiple hardware target, but also in many different (virtualized or not) configurations, and this in an automated way, we implemented *Linux Servers Benchmarking System* (LSBS). LSBS is an open-source tool [18] specifically designed for orchestrating the benchmarking process. LSBS oversees the installation and execution of benchmarks on one or multiple target *platforms* using the aforementioned Ansible playbooks. Once a benchmark has finished, the results are sent back to LSBS which collects them. LSBS takes one or more machines, virtual or not, as targets, with a fresh OS installed on them. Installations and executions of one or more benchmark batches on one or more targets can then be scheduled.

As we must ensure that the software environment is exactly the same each time and can be reproduced, the installation procedure must be extremely precisely defined. The installation of the container runtimes (Docker, k0s) is fully automated and handled by LSBS thru the aforementioned Ansible playbooks. But for installing the base OS (for bare-metal benchmarking), the type 1 hypervisors (hence hypervisors running directly over the metal), and the Ubuntu 22.04 host OS (for VMs), we defined a detailed manual installation procedure, also available in the LSBS repository. In the future, we plan to automate this part as well, using Metal-as-a-Service approaches [15].

2.3 Results collection and visualization system

LSBS provides a webapp front-end, thru which 1) benchmarks runs can be scheduled and monitored (Fig. 1a), 2) raw measurements can be verified (Fig. 2) and 3) comparisons between *platforms* for each hardware and benchmark can be displayed in the form of graphs with built-in statistical tools (Fig 1b). These tools facilitate benchmarking campaigns, for first order result validation notably.



(a) (b)

Fig. 1. LSBS front-end webapp screenshots a) benchmarks scheduling interface. b) results comparison interface, displaying an example result comparing the REST server performance across several *platforms* on the AMD based system (details will be presented below).

Fig. 2. Example of result checking graph for the bare metal deep learning benchmark on the Intel Core i7-4790k hardware, using LSBS visualization features.

3 Results

With our benchmark suite and benchmarking tool LSBS at hand, we conducted performances comparisons on three different hardware systems, which are listed in Table 3. These three systems show some diversity in the hardware (CPU, manufacturer) and belong to different generations.

As for virtualization, to thoroughly compare bare metal with VMs and containers, we thrived to use the most popular hypervisors and container runtimes. For hypervisors, we selected: VMware vSphere Hypervisor 7.0 (ESXi), Microsoft Hyper-V server 2019, Proxmox Virtual Environment 7.3 and XCP-ng 8.2. For containers, Docker 20.10 which uses the runC runtime and k0s, a Kubernetes orchestrator that uses the containerd runtime, in version 1.22.

Table 4 lists the 15 combinations that have been tested on each hardware. Essentially, we test 3 container environments (Docker, k0s, *NONE*) across 5 machines (*bare-metal machine*, Proxmox VM, VMWare VM, XCP-ng VM, Hyper-v VM). Grey lines in Table 4 denote combinations that couldn't be tested on the AMD-B550 based system, due to hardware drivers compatibility issues. For this hardware, we thus have only 9 *platforms* to test. Altogether, we ended up with 15 + 15 + 9 = 39 hardware/virtualization/container combinations - that we call "*platforms*" throughout this document.

Table 3. Systems used for the benchmarks.

CPU	MEMORY	STORAGE	NETWORK
Intel Core i7-4790k 4 cores	24 GB DDR3 1333MHz	1TB SanDisk SSD Plus	HP NC523SFP 10Gb/s
2*Intel Xeon E5-2630 v3 8+8 = 16 cores	128GB DDR4 1866MHz	120 GB Intel SSD DC S3500	Intel X540-AT2 10Gb/s
AMD Ryzen 7 3700X 8 cores	16GB DDR4 3000MHz	1TB SanDisk SSD Plus	HP NC523SFP 10Gb/s

Table 4. Tested container/virtualization combinations. Grey lines have not been tested on the AMD based system

		Bare metal
	Docker over	Bare metal
	K0s over	Bare metal
	Ubuntu Proxmox VM over	Bare metal
Docker over	Ubuntu Proxmox VM over	Bare metal
K0s over	Ubuntu Proxmox VM over	Bare metal
	Ubuntu Hyper-v VM over	Bare metal
Docker over	Ubuntu Hyper-v VM over	Bare metal
K0s over	Ubuntu Hyper-v VM over	Bare metal
	Ubuntu XCP-ng VM over	Bare metal
Docker over	Ubuntu XCP-ng VM over	Bare metal
K0s over	Ubuntu XCP-ng VM over	Bare metal
	Ubuntu VMWare VM over	Bare metal
Docker over	Ubuntu VMWare VM over	Bare metal
K0s over	Ubuntu VMWare VM over	Bare metal

As we have 39 *platforms* across the three hardware and given that the benchmarking suite contains 20 benchmarks, we have scheduled 780 batches. Knowing that we asked LSBS to repeat each execution 20 times within a batch, we thus should have collected 39 x 20 benchmarks x 20 repetitions = 15'600 measurements. In practice, a few batches have failed, resulting in a slightly lower number of measurements.

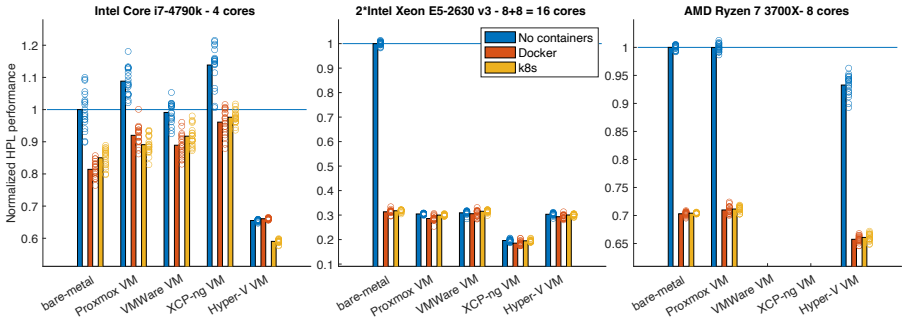
The first phase of result analysis consisted of verifying the consistency of the outcomes across the 20 tests that comprise a single run. This has been done by manually inspecting each of the 39 X 20 = 780 runs on a graph provided by the webapp, as the one visible in Figure 2. We checked for anomalies, for example an overheating of the CPU with scores that decrease after a while or worse scores at the beginning of the run because other operations could still be running in the background. Upon thorough examination, no evident aberrations have been detected in the gathered data (to the exception of the few missing batches).

Next, we computed, for each benchmark, a reference mark by calculating the mean performance of the workload on the "bare-metal" *platform*, using no virtualization nor containers. Then we normalized our results by this mark. Figure 3 show the result for 3 benchmarks: HPL, database experiment and deep-learning experiment. For HPL (Fig. 3a – higher is better), on the Intel Core i7 computer, performance is rather homogenous, but we see hypervisors outperforming bare-metal (best: XCP-ng with 113% performance of bare-metal), to the exception of Hyper-V. Introducing OS virtualization induces a 10-20% overhead, except on Hyper-V. On the dual Xeon hardware, virtualization induces a massive performance drop. Further investigation of these results are required, but this could be due to the fact that Hypervisors are agnostic to NUMA effects of multi-socket systems.

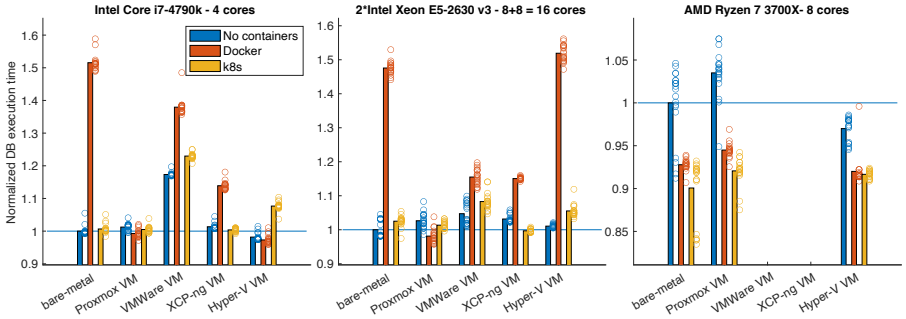
For database results (Fig 3b – lower is better), we note that the results are even more homogeneous, except some outlier *platforms* using Docker. On the AMD system, containerized workloads systematically outperform the bare-metal case.

For deep-learning results (Fig 3c – lower is better), we note again better performances for containerized workloads. We also note a severe performance drop when using Hyper-V on the Intel Core i7 hardware.

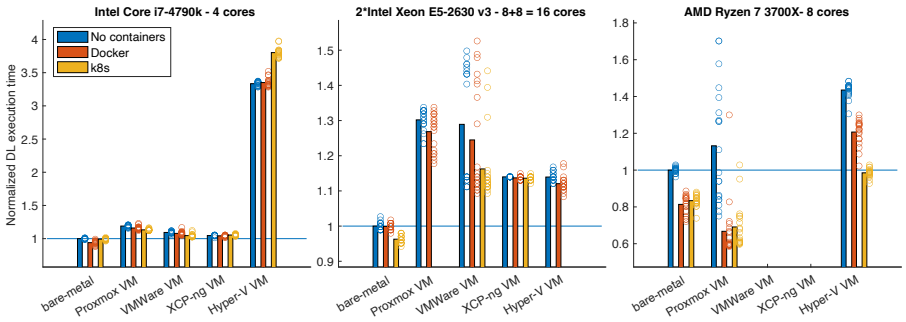
In general, we observe that the performance of hypervisor or container runtimes can significantly vary depending on the hardware.



(a) HPL – higher is better



(b) Database – lower is better



(b) Deep learning – lower is better

Fig. 3. Normalized performance measurement for three benchmarks (a,b,c) across the three hardwares (columns). For HPL (a), higher performance is better, while for database (b) and deep-learning (c), lower execution time is better. Circles denote individual measurements.

Table 5. Hypervisor comparison.

Benchmark	Proxmox	VMWare*	Hyper-V	XCP-ng*	Best
CPU	**0.0119**	0.0112	0.0107	*0.0063*	Proxmox
Memory	9.62	9.87	**10.82**	*6.71*	Hyper-V
Storage	35332	*26894*	29827	**36454**	XCP-ng
Network	**167.43**	*150.31*	159.83	163.42	Proxmox
Blender	**36.33**	42.25	*44.33*	41.50	Proxmox
Deep learning	203.19	200.75	*394.69*	**189.00**	XCP-ng
Database	609	*645*	**587**	605	Hyper-V
REST server	*16.5*	15.5	13.5	**13.0**	XCP-ng

Table 6. Container runtimes comparison.

Benchmark	Docker	Containerd	Best
CPU	0.0097	**0.0098**	Containerd
Memory	**7.67**	7.66	Docker
Storage	**38096**	34592	Docker
Network	149.11	**152.84**	Containerd
Blender	39.33	**39.00**	Containerd
Deep learning	**158.83**	162.00	Docker
Database	773.33	**584.33**	Containerd
REST server	11.33	11.33	--

Table 7. Performance availability with and without virtualization

Benchmark	Bare metal	Container	Vm
CPU	100%	28-86%	28-108%
Memory	100%	61-91%	51-139%
Storage	100%	64-118%	93-197%
Network	100%	95-112%	90-103%
Blender	100%	96-105%	105-112%
Deep learning	100%	96-110%	93-100%
Database	100%	101-117%	69-96%
Rest server	100%	107-130%	90-93%

Next, we looked if an hypervisor clearly dominates the others. For that, we aggregated the performance along each benchmark category (except for applications, which we kept individually) and across the different hardwares, without the use of containers. Results are presented in Table 5. We note that no hypervisor seems to clearly emerge once results are averaged over the different hardwares. Each hypervisor is trailing in at least one category (*italic* figures) and only VMWare never achieve a best score (**bold** figures). We performed the same analysis for container runtimes, whose results are visible in Table 6. Here as well, there is no clear winner.

We further aggregated our results to allow a comparison with our initial data extracted from literature (Table 1). Since we concluded that there is no obvious choice in terms of hypervisor or container runtime for comparing performance with bare-metal, we cherry-picked the best hypervisor and the best container runtime of each category

according to Tables 5 and 6, and finally we extracted the performances using the worst and best hardware. Results are visible on Table 7.

As we noted before, pure CPU performances seems to suffer a lot from NUMA effects, potentially explaining the poor worst performances. We note, however, that as we move toward higher level workloads, these penalties tend to disappear. The storage heavily depends on the hypervisor caching methods. For example, Hyper-V offers impressive random performances but has a massive loss on sequential accesses. Proxmox, with the default caching, offers a less impressive random performance boost but has only a maximum 7% loss on sequential operations. On the network side, while containers and VMs both suffer a 0.1ms ping time increase, the speeds heavily depend on the network card drivers. On intel hardwares, the Hyper-V driver has a big performance loss. The strange UDP behavior observed in [9] was not present in our results. Finally, when running "typical" applications, there is not much difference between bare metal, containers, and virtual machines.

Our final analysis consisted of checking whether virtualization overheads are cumulative: if running a workload in a VM reveals a normalized performance of, say, 90%, and running the same workload in a container returns a normalized performance of 85%, can we expect the performance of a container running in a VM to be 85% * 90% = 76.5%? Figure 4 exhibits this analysis for a couple of *platform*-benchmark pairs. We see that for HPL on the intel Core i7 hardware (top-left – higher is better), the best combination is XCP-ng + k8s. The predictions are also rather accurate, except may be for Hyper-V & Docker.

For deep learning, again on the Intel Core i7 hardware (top-right – less is better), Hyper-V performance is several degraded, and also more degraded than what one could predict. Finally, for database benchmarks on the two Intel based hardwares (bottom – less is better), we see that the predictor works rather well for k8s but doesn't for Docker. This could indicate that with Docker we are not facing a linear overhead, but rather a sort of impedance mismatch, possibly stemming from a misconfiguration of the engine.

4 Discussion

Within the review process, this paper has received many comments and constructive criticism from the reviewers. Here we discuss some of these points.

One criticism was that our benchmark results, which considered *platforms* as "black-boxes" with default and non-optimized settings, do not provide very useful insights on these products and underlying virtualization technologies. We agree that our results are not insightful if one looks for the *intrinsic* performance of a *platform* for executing a particular task. Yet, one of the biggest advantages of virtualization being to reduce the number of physical servers needed in an IT system, in practice one can hardly optimize a *platform* for a specific type of task. In addition, many SMEs operating a private cloud can hardly afford an expert dedicating his time to fine-tune the hypervisor settings depending on the current workloads. Therefore, we do see value in sticking to default, generic hypervisor configuration as tuned by the vendor.

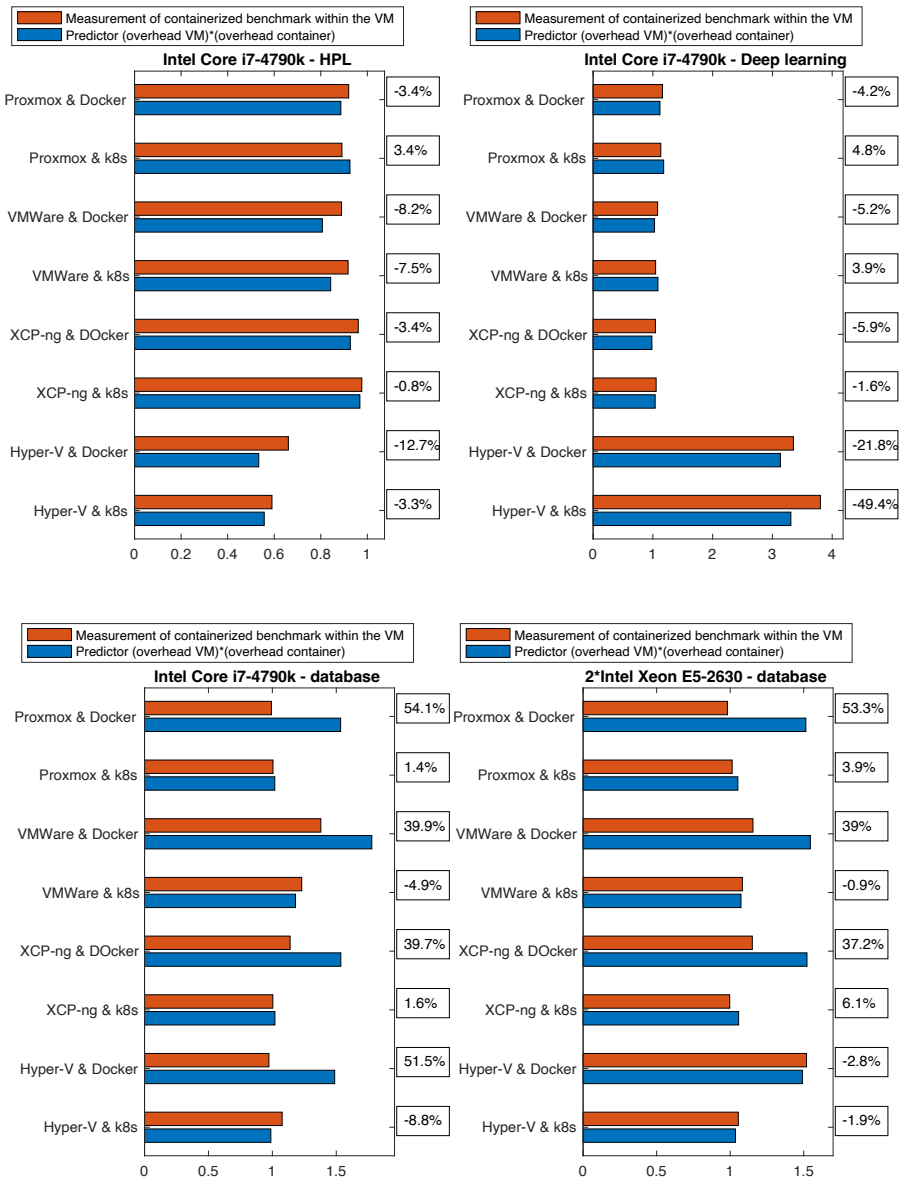


Fig. 4. Comparison of mean performances normalized to the bare-metal case of different VM and container combinations (bar labels), for 4 different *platform*-benchmark pairs. Red bars denote actual performance. Blue bars show a "prediction" obtained by multiplying individual virtualized and containerized performances. Boxes indicate the error of the prediction.

Other remarks mentioned that we should have disclosed many more settings as the BIOS configuration flags, hypervisors settings, and offered a more in-depth description of the test setups. We can clarify as follows: no particular configurations have been applied to the target BIOSs except disabling the power save modes, which means we sticked to default settings. All the hypervisors have been installed with the default settings as described in the GitHub project [18]. For sure there are many parameters to describe, and to play with, but we believe it is also interesting to report results taken "in the wild". And, again, most users are unaware of all these parameters.

Specifically, we've been asked if direct assignment or paravirtualized I/O were used. While direct assignment or passthrough works great in a specialized cluster, it is difficult to use in a typical general purpose virtualization cluster, so all our VMs are configured with paravirtualized I/O devices. In each case, the maximum available vCPUs and memory is allocated to the VM. The installation and configuration are described in the Github project [18].

We been rightfully told that both Docker and k0s use containerd and runC. However, there is an added layer, the container runtime interface between k0s and containerd [19].

Finally, a reviewer has been extremely surprised to see notably high overheads from running HPL in a container (Fig 3a) and hinted a misconfiguration of the container engine. Yet the default configuration has been used. It is possible that the container engine is, by default, optimized not for the performance but to work as good as possible in a maximum of different environment. It could be interesting to conduct a study on how to automatically optimize this configuration or at least notify and expose the problem to the user.

Generally, our study opens the door to many further explorations.

5 Conclusions and Perspectives

The focus of this research was to quantify the performance losses introduced by virtualization to provide valuable information that can help IT departments choose the most efficient technology for their needs: keep using bare-metal machine to guarantee performances, or shift to VMs, containers, or a mix thereof.

A large benchmarking system was created and tested on hardware from different eras and manufacturers, yielding over 15,000 results to compare the performances of bare metal mixed with 4 hypervisors and 2 different container runtimes.

The results showed that by choosing the right virtualization technologies, it is possible to minimize losses to 5% for a container and 10% for a VM. Users should nevertheless be careful to NUMA effects for CPU intensive tasks. Depending on the task, an hypervisor can make a better usage of the available resources. In many cases, VMs outperformed the bare-metal case, even in terms of CPU. This interesting fact deserves further investigation. Both container runtimes and hypervisors offer caching mechanisms that can be interesting to use on storage intensive tasks. Since the performances vary across hypervisors and container runtimes, it might thus be interesting to conduct *in-situ* lab benchmarking to make the right virtualization choices.

Generally we thus conclude that virtualization (VM or OS) does not kills performance, at least when the hardware is not shared among multiple VMs and/or containers.

However, our results show that there can be adversarial situations where performance is highly degraded. These situations can potentially be alleviated by changing configuration flags, yet it might be hard to detect these situations. And changing the configuration to solve one situation might affect other situations.

It is also important to note that we *only* considered single workload performance measurements. The performance evaluation when multiple virtualized workloads compete for the resources is kept for future work.

In the future, we also plan to regularly replicate these benchmarks on newer hardware and software systems as it was found that the figures, especially for storage IO, have changed significantly over the years. It would also be interesting to see if and where software developers choose to make improvements and if all the features added each year do not result in increased overhead.

Eventually, we plan to investigate if, when nesting multiple levels of VMs, the losses increase according to the number of added layer or if only the lowest level one matters.

References

1. A. Gavrilovska, et al.: High-Performance Hypervisor Architectures: Virtualization in HPC Systems, *HPCVirt 07*, (2007).
2. J. White, et al.: A Survey of Virtualization Technologies With Performance Testing, *CoRR, vol. abs/1010.3233. Available: http://arxiv.org/abs/1010.3233* (2010).
3. P. Padala, et al.: Performance Evaluation of Virtualization for Server Consolidation, *HP Laboratories Report NO. HPL-2007–59R1* (2007).
4. M. G. Xavier, et al.: Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments, *Conf. on Par., Distr., and Network-Based Proc.*, doi: 10.1109/PDP.2013.41 (2013)
5. S. A. Babu, et al. System Performance Evaluation of Para Virtualization, Container Virtualization, and Full Virtualization Using Xen, OpenVZ, and XenServer, *Int. Conference on Advances in Computing and Communications*, doi: 10.1109/ICACC.2014.66 (2014)
6. C. Arango, et al.: Performance Evaluation of Container-based Virtualization for High Performance Computing Environments, *https://doi.org/10.48550/arXiv.1709.1014*0 (2017)
7. R. McDougall et al.: Virtualization performance: perspectives and challenges ahead. *SIGOPS Oper. Syst. Rev. 44, 4* (2010)
8. Z. Li, et al.: Performance Overhead Comparison between Hypervisor and Container Based Virtualization, *IEEE Int. Conf. on Adv. Inf. Net. and Appl. (AINA)*, doi: 10.1109/AINA.2017.79 (2017)
9. R. Morabito, et al.: Hypervisors vs. Lightweight Virtualization: A Performance Comparison, *IEEE Int. Conf. on Cloud Eng*., doi: 10.1109/IC2E.2015.74 (2015)
10. S. Shirinbab, et al.: Performance evaluation of container and virtual machine running cassandra workload, *Int. Conf. of Cloud Comp. Tech. and Appl. (CloudTech)*, doi: 10.1109/CloudTech.2017.8284700 (2017).
11. S. Shirinbab, et al.: Performance Comparison of KVM, VMware and XenServer using a Large Telecommunication Application, *CCGrid* (2014).
12. *https://www.proxmox.com/en/news/press-releases/proxmox-ve-3-4-released*
13. *https://www.docker.com/blog/speed-boost-achievement-unlocked-on-docker-desktop-4-6-for-mac/*
14. *https://www.ansible.com*
15. *http://mass.io*

16. D. Tres, Switchkaas factsheet, *https://www.switch.ch/export/sites/default/kubernetes-as-a-service/.galleries/files/SWITCHkaas-Factsheet-EN.pdf*
17. J. T. Lim, et al.. NEVE: Nested Virtualization Extensions for ARM. *Symposium on Operating Systems Principles* (2017).
18. Github.com/jojoc4/LSBS
19. *https://www.techtarget.com/searchitoperations/tip/A-breakdown-of-container-runtimes-for-Kubernetes-and-Docker#:~:text=The%20container%20runtime%20is%20the,OCI%2Dcompliant%20runtime%20should%20work.*