

Performance Overhead Comparison between Hypervisor and Container based Virtualization

Zheng Li*, Maria Kihl*, Qinghua Lu[†] and Jens A. Andersson*

*Department of Electrical and Information Technology, Lund University, Lund, Sweden

Email: {zheng.li, maria.kihl, jens_a.andersson}@eit.lth.se

[†]College of Computer and Communication Engineering, China University of Petroleum, Qingdao, China

Email: dr.qinghua.lu@gmail.com

Abstract—The current virtualization solution in the Cloud widely relies on hypervisor-based technologies. Along with the recent popularity of Docker, the container-based virtualization starts receiving more attention for being a promising alternative. Since both of the virtualization solutions are not resource-free, their performance overheads would lead to negative impacts on the quality of Cloud services. To help fundamentally understand the performance difference between these two types of virtualization solutions, we use a physical machine with “just-enough” resource as a baseline to investigate the performance overhead of a standalone Docker container against a standalone virtual machine (VM). With findings contrary to the related work, our evaluation results show that the virtualization’s performance overhead could vary not only on a feature-by-feature basis but also on a job-to-job basis. Although the container-based solution is undoubtedly lightweight, the hypervisor-based technology does not come with higher performance overhead in every case. For example, Docker containers particularly exhibit lower QoS in terms of storage transaction speed.

Index Terms—Cloud Service; Container; Hypervisor; Performance Overhead; Virtualization Technology

1. Introduction

As a key element of Cloud computing, virtualization plays various vital roles in supporting Cloud services, ranging from resource isolation to resource provisioning. The existing virtualization technologies can roughly be distinguished between the hypervisor-based and the container-based solutions. Considering their own resource consumption, both virtualization solutions inevitably introduce performance overheads when offering Cloud services, and the performance overheads could then lead to negative impacts to the corresponding quality of service (QoS). Therefore, it would be crucial for both Cloud providers (e.g., for improving infrastructural efficiency) and consumers (e.g., for selecting services wisely) to understand to what extent a candidate virtualization solution incurs influence on the Cloud’s QoS.

Given the characteristics of these two virtualization solutions (cf. the background in Section 2), an intuitive hy-

pothesis could be: *a container-based service exhibits better performance than its corresponding hypervisor-based VM service*. Nevertheless, there is little quantitative evidence to help test this hypothesis in an “apple-to-apple” manner, except for those qualitative discussions. Therefore, we decided to use a physical machine with “just-enough” resource as a baseline to quantitatively investigate and compare the performance overheads between the container-based and hypervisor-based virtualizations. In particular, since Docker is currently the most popular container solution [1] and VMWare is one of the leaders in the hypervisor market [2], we chose Docker and VMWare Workstation 12 Pro to represent the two virtualization solutions respectively.

According to the clarifications in [3], our qualitative investigations can be regulated by the discipline of experimental computer science (ECS). By employing ECS’s recently available Domain Knowledge-driven Methodology (DoKnowMe) [4], we experimentally explored the performance overheads of different virtualization solutions on a feature-by-feature basis.

The experimental results and analyses show that the aforementioned hypothesis is not true in all the cases. For example, we do not observe computation performance difference between those service types with respect to solving a combinatorially hard chess problem; and the container even leads to higher storage performance overhead than the VM when reading/writing data byte by byte. Moreover, we find that the remarkable performance loss incurred by both virtualization solutions usually appears in the performance variability. Overall, the contributions of this work are three-fold:

- Our experimental results and analyses can help both researchers and practitioners to better understand the fundamental performance of the container-based and hypervisor-based virtualization technologies. It is notable that the performance engineering in ECS can roughly be distinguished between two stages: the first stage is to reveal the primary performance of specific (system) features, while the second stage is generally based on the first-stage evaluation to investigate real-world application cases. Thus, this work can be viewed as a foundation for more sophisticated evaluation studies in the future.

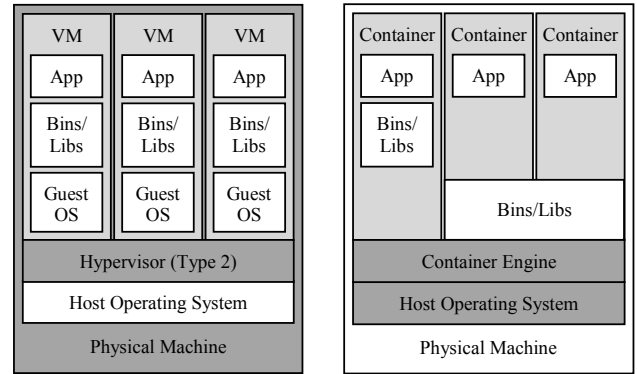
- Our method of calculating performance overhead can easily be applied or adapted to different evaluation scenarios by others. The literature shows that the “performance overhead” has normally been used in the context of qualitative discussions. By quantifying such an indicator, our study essentially provides a concrete lens into the case of performance comparisons.
- The whole evaluation logic and details reported in this paper can be viewed as a reusable and traceable template for evaluating Docker containers. Since the Docker project is still quickly growing [5], the evaluation results could gradually be out of date. Benefiting from this template, future evaluations can conveniently be repeated or replicated by different evaluators at different times and locations.

The remainder of this paper is organized as follows. Section 2 summarizes the background knowledge of container-based and the hypervisor-based virtualizations and highlights the existing work related to their performance comparisons. Section 3 introduces the performance evaluation implementation including the methodology employed in our study. The detailed performance overhead investigation is divided into two reporting parts, namely pre-experimental activities and experimental results & analyses, and they are correspondingly described into Section 3.2 and 3.3 respectively. Conclusions and some future work are discussed in Section 4.

2. Background and Related Work

When it comes to the Cloud virtualization, the de facto solution is to employ the hypervisor-based technologies, and the most representative Cloud service type is offering virtual machines (VMs) [6]. In this virtualization solution, the hypervisor manages physical computing resources and makes isolated slices of hardware available for creating VMs [5]. We can further distinguish between two types of hypervisors, namely the bare-metal hypervisor that is installed directly onto the computing hardware, and the hosted hypervisor that requires a host operating system (OS). To make a better contrast between the hypervisor-related and container-related concepts, we particularly emphasize the second hypervisor type, as shown in Figure 1a. Since the hypervisor-based virtualization provides access to physical hardware only, each VM needs a complete implementation of a guest OS including the binaries and libraries necessary for applications [7]. As a result, the guest OS will inevitably incur resource competition against the applications running on the VM service, and essentially downgrade the QoS from the application’s perspective.

To relieve the performance overhead of hypervisor-based virtualization, researchers and practitioners recently started promoting an alternative and lightweight solution, namely container-based virtualization. In fact, the foundation of the container technology can be traced back to the Unix `chroot` command in 1979 [7], while this technology is eventually evolved into virtualization mechanisms like Linux VServer, OpenVZ and Linux Containers (LXC) along



(a) Hypervisor-based virtual service. (b) Container-based virtual service.

Figure 1. Different architectures of hypervisor-based and container-based virtual services.

with the booming of Linux [8]. Unlike the hardware-level solution of hypervisors, containers realize virtualization at the OS level and utilize isolated slices of the host OS to shield their contained applications [7]. In essence, a container is composed of one or more lightweight images, and each image is a prebaked and replaceable file system that includes necessary binaries, libraries or middlewares for running the application. In the case of multiple images, the read-only supporting file systems are stacked on top of each other to cater for the writable top-layer file system [1]. With this mechanism, as shown in Figure 1b, containers enable applications to share the same OS and even binaries/libraries when appropriate. As such, compared to VMs, containers would be more resource efficient by excluding the execution of hypervisor and guest OS, and more time efficient by avoiding booting (and shutting down) a whole OS [5], [9]. Nevertheless, it has been identified that the cascading layers of container images come with inherent complexity and performance penalty [10]. In other words, the container-based virtualization technology could also negatively impact the corresponding QoS due to its performance overhead.

Although the performance advantage of containers were investigated in several pioneer studies [2], [8], [11], the container-based virtualization solution did not gain significant popularity until the recent underlying improvements in the Linux kernel, and especially until the emergence of Docker [12]. Starting from an open-source project in early 2013 [5], Docker quickly becomes the most popular container solution [1] by significantly facilitating the management of containers. Technically, through offering the unified tool set and API, Docker relieves the complexity of utilizing the relevant kernel-level techniques including the LXC, the cgroup and a copy-on-write filesystem. To examine the performance of Docker containers, a molecular modeling simulation software [13] and a PostgreSQL database-based Joomla application [14] have been used to benchmark the Docker environment against the VM environment.

The closest work to ours is the CPU-oriented study [15] and the IBM research report [16] on the performance

comparison of VM and Linux containers. However, both studies are incomplete (e.g., the former was not concerned with the non-CPU features, and the latter did not finish the container’s network evaluation). More importantly, our work denies the IBM report’s finding that “containers and VMs impose almost no overhead on CPU and memory usage” and also doubts about “Docker equals or exceeds KVM performance in every case”. Furthermore, in addition to the average performance overhead of virtualization technologies, we are more concerned with their overhead in performance variability.

Note that, although there are also performance studies on deploying containers inside VMs (e.g., [17], [18]), such a redundant structure might not be suitable for an “apple-to-apple” comparison between Docker containers and VMs, and thus we do not include this virtualization scenario in this study.

3. Performance Evaluation Implementation

3.1. Performance Evaluation Methodology

Since the comparison between the container’s and the VM’s performance overheads is essentially based on their performance evaluation, we define our work as a performance evaluation study that belongs to the field of ECS [3]. Considering that “evaluation methodology underpins all innovation in experimental computer science” [19], we employ the methodology DoKnowMe [4] to guide evaluation implementations in this study. DoKnowMe is an abstract evaluation methodology on the analogy of “class” in object-oriented programming. By integrating domain-specific knowledge artefacts, DoKnowMe can be customized into specific methodologies (by analogy of “object”) to facilitate evaluating different concrete computing systems. To better structure our report, we divide our DoKnowMe-driven evaluation implementation into pre-experimental activities (cf. Section 3.2) and experimental results & analyses (cf. Section 3.3).

3.2. Pre-Experimental Activities

3.2.1. Requirement Recognition. Following DoKnowMe, the whole evaluation implementation is essentially driven by the recognized requirements. In general, the requirement recognition is to define a set of specific requirement questions both to facilitate understanding the real-world problem and to help achieve clear statements of the corresponding evaluation purpose. In this case, the basic requirement is to give a fundamental quantitative comparison between the hypervisor-based and the container-based virtualization solutions. Since we concretize these two virtualization solutions into VMWare Workstation VMs and Docker containers respectively, such a requirement can further be specified into two questions:

RQ1: How much performance overhead does a standalone Docker container introduce over its base physical machine?

TABLE 1. METRICS AND BENCHMARKS FOR THIS EVALUATION STUDY

Physical Property	Capacity Metric	Benchmark	Version
Communication	Data Throughput	Iperf	2.0.5
Computation	(Latency) Score	HardInfo	0.5.1
Memory	Data Throughput	STREAM	5.10
Storage	Transaction Speed	Bonnie++	1.97.1
Storage	Data Throughput	Bonnie++	1.97.1

RQ2: How much performance overhead does a standalone VM introduce over its base physical machine?

Considering that virtualization technologies could result in service performance variation [20], we are also concerned with the container’s and VM’s potential variability overhead besides their average performance overhead:

RQ3: How much performance variability overhead does a standalone Docker container introduce over its base physical machine during a particular period of time?

RQ4: How much performance variability overhead does a standalone VM introduce over its base physical machine during a particular period of time?

3.2.2. Service Feature Identification. Recall that we treat Docker containers as an alternative type of Cloud service to VMs. By using the taxonomy of Cloud services evaluation [21], we examine the communication-, computation-, memory- and storage-related QoS aspects; and then we focus on the service features including communication data throughput, computation latency, memory data throughput, and storage transaction speed and data throughput.

3.2.3. Metrics/Benchmarks Listing and Selection. The selection of evaluation metrics usually depends on the availability of benchmarks. According to our previous experience of Cloud services evaluation, we choose relatively lightweight and popular benchmarks to try to minimize the potential benchmarking bias, as listed in Table 1. For example, Iperf has been identified to be able to deliver more precise results by consuming less system resources. In fact, except for STREAM that is the de facto memory evaluation benchmark included in the HPC Challenge Benchmark (HPCC) suite, the other benchmarks are all Ubuntu’s built-in utilities.

In particular, although Bonnie++ only measures the amount of data processed per second, the disk I/O transactions are on a byte-by-byte basis when accessing small size of data. Therefore, we consider to measure storage transaction speed when operating byte-size data and measure storage data throughput when operating block-size data. As for the property computation, considering the diversity in CPU jobs (e.g., integer and floating-point calculations), we employ HardInfo that includes six micro-benchmarks to generate performance scores.

When it comes to the performance overhead, we use the business domain's *Overhead Ratio*¹ as an analogy to its measurement. In detail, we treat the performance loss compared to a baseline as the expense, while imagining the baseline performance to be the overall income, as defined in Equation (1).

$$O_p = \frac{|P_m - P_b|}{P_b} \times 100\% \quad (1)$$

where O_p refers to the performance overhead; P_m denotes the benchmarking result as a measurement of a service feature; P_b indicates the baseline performance of the service feature; and then $|P_m - P_b|$ represents the corresponding performance loss. Note that the physical machine's performance is used as the baseline in our study. Moreover, considering possible observational errors, we allow a margin of error for the confidence level as high as 99% with regarding to the benchmarking results. In other words, we will ignore the difference between the measured performance and its baseline if the calculated performance overhead is less than 1% (i.e. if $O_p < 1\%$, then $P_m = P_b$).

3.2.4. Experimental Factor Listing and Selection. The identification of experimental factors plays a prerequisite role in the following experimental design. More importantly, specifying the relevant factors would be necessary for improving the repeatability of experimental implementations. By referring to the experimental factor framework of Cloud services evaluation [22], we choose the resource- and workload-related factors as follows. In particular, considering the possible performance overhead compensation from powerful computing resources, we try to stress the experimental condition by employing a “just-enough” testbed.

The resource-related factors:

- *Resource Type*: Given the evaluation requirement, we have essentially considered three types of resources to support the imaginary Cloud service, namely physical machine, container and VM.
- *Communication Scope*: We test the communication between our local machine and an Amazon EC2 t2.micro instance. The local machine is located in our broadband lab at Lund University, and the EC2 instance is from Amazon's available zone ap-southeast-1a within the region Asia Pacific (Singapore).
- *Communication Ethernet Index*: Our local side uses a Gigabit connection to the Internet, while the EC2 instance at remote side has the “Low to Moderate” networking performance defined by Amazon.
- *CPU Index*: Recall that we have employed “just-enough” computing resource. The physical machine's CPU model is chosen to be Intel Core™2 Duo Processor T7500. The processor has two cores with the 64-bit architecture, and its base frequency is 2.2 GHz. We allocate both CPU cores to the standalone VM upon the physical machine.

- *Memory Size*: The physical machine is equipped with a 3GB DDR2 SDRAM. When running the VMWare Workstation Pro without launching any VM, “watch -n 5 free -m” shows a memory usage of 817MB while leaving 2183MB free in the physical machine. Therefore, we set the memory size to 2GB for the VM to avoid (at least to minimize) the possible memory swapping.
- *Storage Size*: There are 120GB of hard disk in the physical machine. Considering the space usage by the host operating system, we allocate 100GB to the VM.
- *Operating System*: Since Docker requires a 64-bit installation and Linux kernels older than 3.10 do not support all the features for running Docker containers, we choose the latest 64-bit Ubuntu 15.10 as the operating system for both the physical machine and the VM. In addition, according to the discussions about base images in the Docker community [23], [24], we intentionally set an OS base image (by specifying FROM ubuntu:15.10 in the Dockerfile) for all the Docker containers in our experiments. Note that a container's OS base image is only a file system representation, while not acting as a guest OS.

The workload-related factors:

- *Duration*: For each evaluation experiment, we decided to take a whole-day observation plus one-hour warming up (i.e. 25 hours).
- *Workload Size*: The experimental workloads are pre-defined by the selected benchmarks. For example, the micro-benchmark CPU Fibonacci generates workload by calculating the 42nd Fibonacci number. In particular, the benchmark Bonnie++ distinguishes between reading/writing byte-size and block-size data.

3.2.5. Experimental Design. It is clear that the identified factors are all with single value except for the *Resource Type*. Therefore, a straightforward design is to run the individual benchmarks on each of the three types of resources independently for a whole day plus one hour.

Furthermore, following the conceptual model of IaaS performance evaluation [25], we record the experimental design into a blueprint both to facilitate our experimental implementations and to help other evaluators replicate/repeat our study. Due to the space limit, we share the experimental blueprint online as a supplementary document.²

3.3. Experimental Results and Analyses

3.3.1. Communication Evaluation Result and Analysis. For the purpose of “apple-to-apple” comparison, we force both the container and the VM to employ Network Address Translation (NAT) to establish outgoing connections. Since they require port binding/forwarding to accept incoming connections, we only test the outgoing communication performance to reduce the possibility of configurational noise,

¹<http://www.investopedia.com/terms/o/overhead-ratio.asp>

²The experimental blueprint is shared online at <https://drive.google.com/file/d/0B9KzcoAAmi43WTFuTXBsZ0NRd1U/>

TABLE 2. COMMUNICATION BENCHMARKING RESULTS USING IPERF

Resource Type	Average	Standard Deviation
Physical machine	29.066 Mbits/sec	1.282 Mbits/sec
Container	28.484 Mbits/sec	1.978 Mbits/sec
Virtual machine	12.843 Mbits/sec	2.979 Mbits/sec

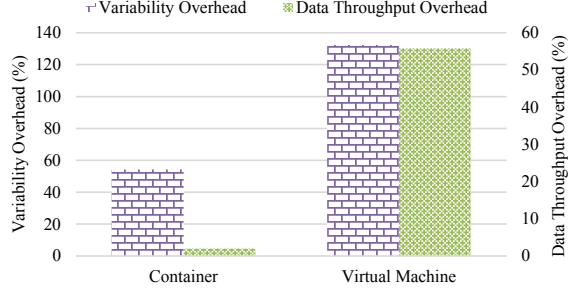


Figure 2. Communication data throughput and its variability overhead of a standalone Docker container vs. VM (using the benchmark Iperf).

by setting the remote EC2 instance to Iperf server and using the local machine, container and VM all as Iperf clients.

The benchmarking results of repeating `iperf -c XXX.XXX.XXX.XXX -t 15` (with a one-minute interval between every two consecutive trials) are listed in Table 2. The XXX.XXX.XXX.XXX denotes the external IP address of the EC2 instance used in our experiments. Note that, unlike the other performance features, the communication data throughput delivers periodical and significant fluctuations, which might be a result from the network resource competition at both our local side and the EC2 side during working hours. Therefore, we particularly focus on the longest period of relatively stable data out of the whole-day observation, and thus the results here are for rough reference only.

Given the extra cost of using the NAT network to send and receive packets, there would be unavoidable performance penalties for both the container and the VM. Using Equation (1), we calculate their communication performance overheads, as illustrated in Figure 2.

A clear trend is that, compared to the VM, the container loses less communication performance, with only 2% data throughput overhead and around 54% variability overhead. However, it is surprising to see a more than 55% data throughput overhead for the VM. Although we have double checked the relevant configuration parameters and redone several rounds of experiments to confirm this phenomenon, we still doubt about the hypervisor-related reason behind such a big performance loss. We particularly highlight this observation to inspire further investigations.

3.3.2. Computation Evaluation Result and Analysis.

Recall that HardInfo’s six micro benchmarks deliver both “higher=better” and “lower=better” CPU scores. To facilitate experimental analysis, we use the two equations below to standardize the “higher=better” and “lower=better” benchmarking results respectively.

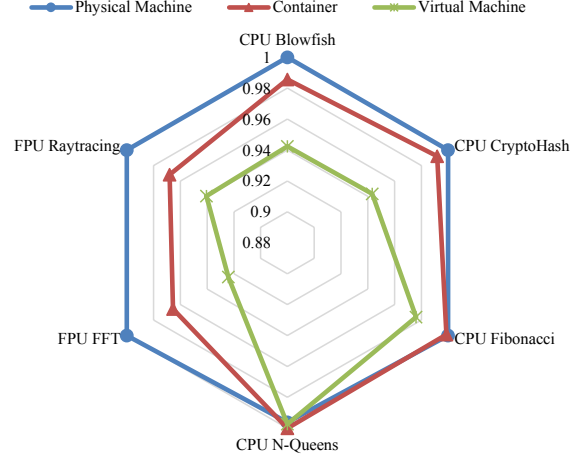


Figure 3. Computation benchmarking results by using HardInfo.

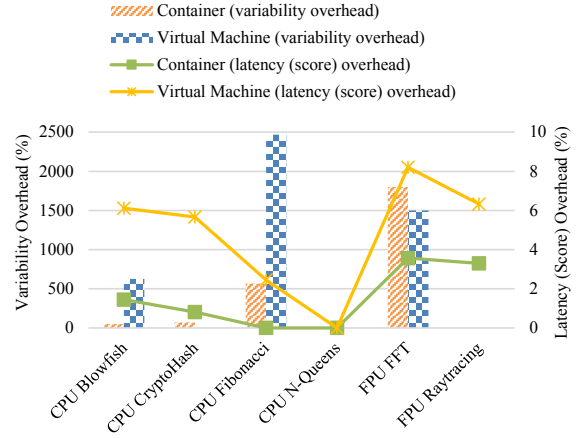


Figure 4. Computation latency (score) and its variability overhead of a standalone Docker container vs. VM (using the tool kit HardInfo).

$$HB_i = \frac{Benchmarking_i}{\max(Benchmarking_{1,2,...,n})} \quad (2)$$

$$LB_i = \frac{1}{\frac{Benchmarking_i}{\max(\frac{1}{Benchmarking_{1,2,...,n}})}} \quad (3)$$

where HB_i further scores the service resource type i by standardizing the “higher=better” benchmarking result $Benchmarking_i$; and similarly, LB_i represents the standardized “lower=better” CPU score of the service resource type i . Note that Equation (3) essentially offers the “lower=better” benchmarking results a “higher=better” representation through reciprocal standardization.

Thus, we can use a radar plot to help intuitively contrast the performance of the three resource types, as demonstrated in Figure 3. For example, the different polygon sizes clearly indicate that the container generally computes faster than the VM, although the performance differences are on a case-by-case basis with respect to different CPU job types.

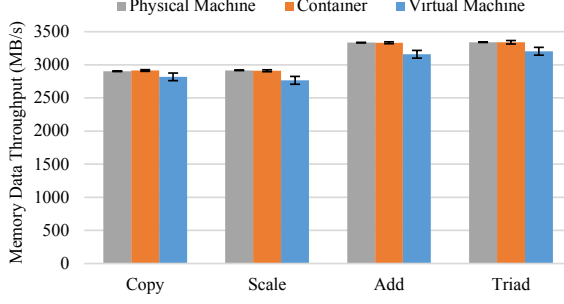


Figure 5. Memory benchmarking results by using STREAM. Error bars indicate the standard deviations of the corresponding memory data throughput.

Nevertheless, our experimental results do not display any general trend in variability of those resources’ computation scores. As can be seen from the calculated performance overheads (cf. Figure 4), the VM does not even show worse variability than the physical machine when running CPU CryptoHash, CPU N-Queens and FPU Raytracing. On the contrary, there is an almost 2500% variability overhead for the VM when calculating the 42nd Fibonacci number. In particular, the virtualization technologies seem to be sensitive to the Fourier transform jobs (the benchmark FPU FFT), because the computation latency overhead and the variability overhead are relatively high for both the container and the VM.

3.3.3. Memory Evaluation Result and Analysis.

STREAM measures sustainable memory data throughput by conducting four typical vector operations, namely Copy, Scale, Add and Triad. We directly visualize the benchmarking results into Figure 5 to facilitate our observation. As the first impression, it seems that the VM has a bit poorer memory data throughput, and there is little difference between the physical machine and the Docker container in the context of running STREAM.

By calculating the performance overhead in terms of memory data throughput and its variability, we are able to see the significant difference among these three types of resources, as illustrated in Figure 6. Take the operation Triad as an example, although the container performs as well as the physical machine on average, the variability overhead of the container is more than 500%; similarly, although the VM’s Triad data throughput overhead is around 4% only, its variability overhead is almost 1400%. In other words, the memory performance loss incurred by both virtualization techniques is mainly embodied with the increase in the performance variability.

In addition, it is also worth notable that the container’s average Copy data throughput is even slightly higher than the physical machine (i.e. 2914.023MB/s vs. 2902.685MB/s) in our experiments. Recall that we have considered a 1% margin of error. Since those two values are close to each other within this error margin, here we ignore such an irregular phenomenon as an observational error.

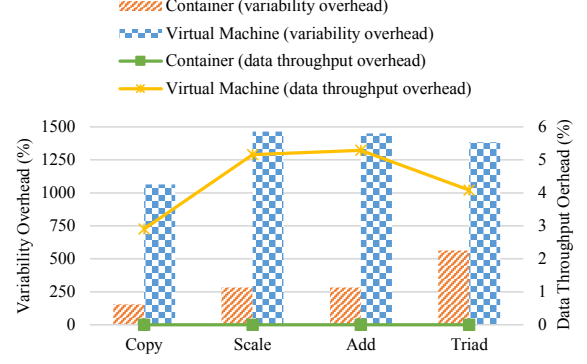


Figure 6. Memory data throughput and its variability overhead of a standalone Docker container vs. VM (using the benchmark STREAM).

3.3.4. Storage Evaluation Result and Analysis. For the test of disk reading and writing, Bonnie++ creates a dataset twice the size of the involved RAM memory. Since the VM is allocated 2GB of RAM, we also restrict the memory usage to 2GB for Bonnie++ on both the physical machine and the container, by running “sudo bonnie++ -r 2048 -n 128 -d / -u root”. Correspondingly, the benchmarking trials are conducted with 4GB of random data on the disk. When Bonnie++ is running, it carries out various storage operations ranging from data reading/writing to file creating/deleting. Here we only focus on the performance of reading/writing byte- and block-size data.

To help highlight several different observations, we plot the trajectory of the experimental results along the trial sequence during the whole day, as shown in Figure 7. The first surprising observation is that, all the three resource types have regular patterns of performance jitter in block writing, rewriting and reading. Due to the space limit, we do not report their block rewriting performance in this paper. By exploring the hardware information, we identified the hard disk drive (HDD) model to be ATA Hitachi HTS54161, and its specification describes “It stores 512 bytes per sector and uses four data heads to read the data from two platters, rotating at 5,400 revolutions per minute”. As we know, the hard disk surface is divided into a set of concentric circular tracks. Given the same rotational speed of an HDD, the outer tracks would have higher data throughput than the inner ones. As such, those regular patterns might indicate that the HDD heads sequentially shuttle between outer and inner tracks when consecutively writing/reading block data during the experiments.

The second surprising observation is that, unlike most cases in which the VM has the worst performance, the container seems significantly poor at accessing the byte size of data, although its performance variability is clearly the smallest. We further calculate the storage performance overhead to deliver more specific comparison between the container and the VM, and draw the results into Figure 8. Note that, in the case when the container’s/VM’s variability is smaller than the physical machine’s, we directly set the corresponding variability overhead to zero rather than

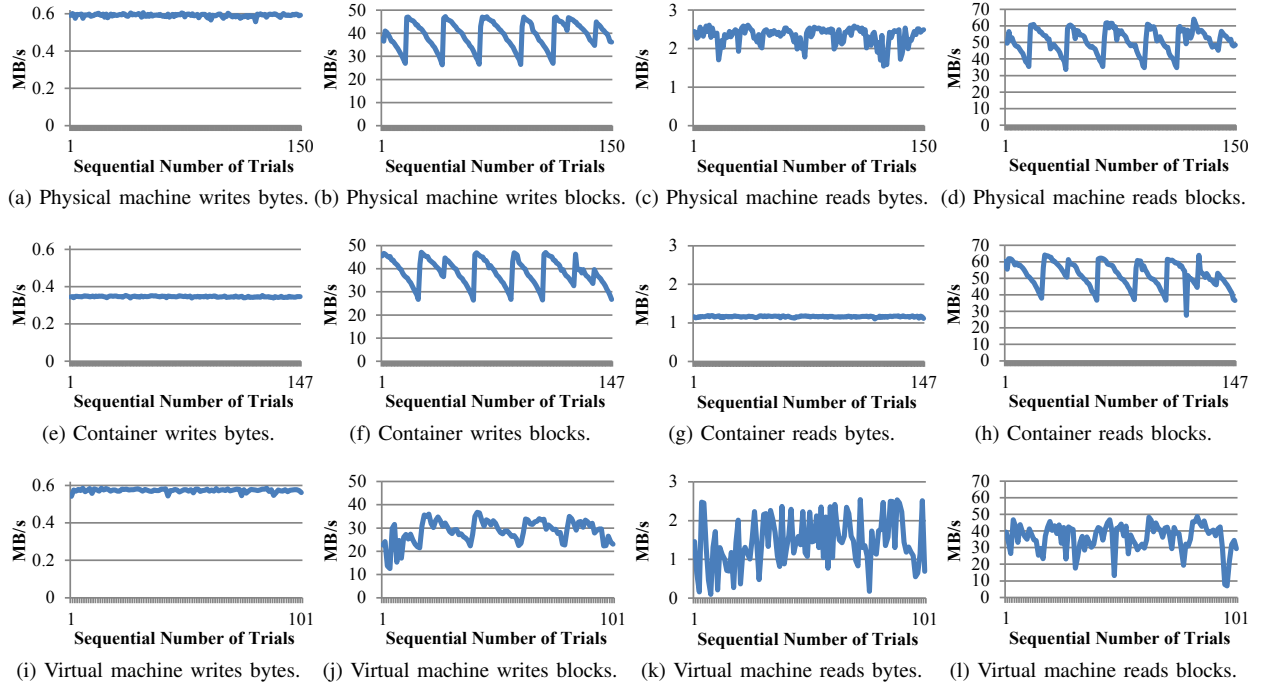


Figure 7. Storage benchmarking results by using Bonnie++ during 24 hours. The maximum x-axis scale indicates the iteration number of the Bonnie++ test (i.e. the physical machine, the container and the VM run 150, 147 and 101 tests respectively).

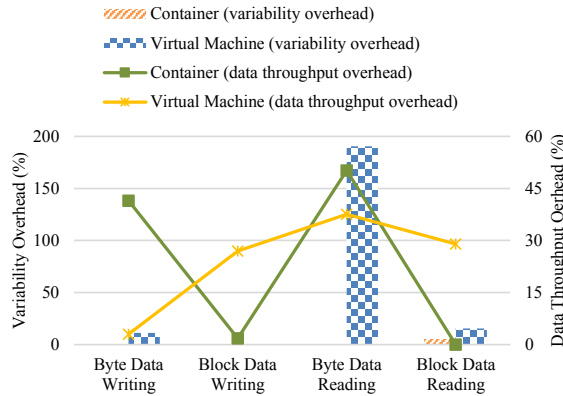


Figure 8. Storage data throughput and its variability overhead of a standalone Docker container vs. VM (using the benchmark Bonnie++).

allowing any performance overhead to be negative. Then, the bars in the chart indicate that the storage variability overheads of both virtualization technologies are nearly negligible except for reading byte-size data on the VM (up to nearly 200%). Although the storage driver is a known bottleneck for a container's internal disk I/O, it is still surprising that the container brings around 40% to 50% data throughput overhead when performing disk operations on a byte-by-byte basis. In other words, the container solution might not be suitable for the Cloud applications that have frequent and unpredictable intermediate data generation and consumption. On the contrary, there is relatively trivial performance loss in VM's byte data writing. However, the

VM has roughly 30% data throughput overhead in other disk I/O scenarios, whereas the container barely incurs overhead when reading/writing large size of data.

Our third observation is that, the storage performance overhead of different virtualization technologies can also be reflected through the total number of the iterative Bonnie++ trials. As pointed by the maximum x-axis scale in Figure 7, the physical machine, the container and the VM can respectively finish 150, 147 and 101 rounds of disk tests during 24 hours. Given this information, we estimate the container's and the VM's storage performance overhead to be 2% ($= |147 - 150| / 150$) and 32.67% ($= |101 - 150| / 150$) respectively.

4. Conclusion

Following the performance evaluation methodology DoKnowMe, we draw conclusions mainly by answering the predefined requirement questions. Driven by RQ1 and RQ2, our evaluation result largely confirms the aforementioned qualitative discussions: The container's average performance is generally better than the VM's and is even comparable to that of the physical machine with regarding to many features. Specifically, the container has less than 4% performance overhead in terms of communication data throughput, computation latency, memory data throughput and storage data throughput. Nevertheless, the container-based virtualization could hit a bottleneck of storage transaction speed, with the overhead up to 50%. Note that, as mentioned previously, we interpret the byte-size data throughput into

storage transaction speed, because each byte essentially calls a disk transaction here. In contrast, although the VM delivers the worst performance in most cases, it could perform as well as the physical machine when solving the N-Queens problem or writing small-size data to the disk.

Driven by RQ3 and RQ4, we find that the performance loss resulting from virtualizations is more visible in the performance variability. For example, the container's variability overhead could reach as high as over 500% with respect to the Fibonacci calculation and the memory Triad operation. Similarly, although the container generally shows less performance variability than the VM, there are still exceptional cases: The container has the largest performance variation in the job of computing Fourier transform, whereas even the VM's performance variability is not worse than the physical machine's when running CryptoHash, N-Queens, and Raytracing jobs.

Overall, our work reveals that the performance overheads of these two virtualization technologies could vary not only on a feature-by-feature basis but also on a job-to-job basis. Although the container-based solution is undoubtedly lightweight, the hypervisor-based technology does not come with higher performance overhead in every case. Based on such a fundamental evaluation study, we will gradually apply Docker containers to different real-world applications in the coming future. The application-oriented practices will also be replicated in the hypervisor-based virtual environment for further comparison case studies.

Acknowledgment

This work is supported by the Swedish Research Council (VR) for the project "Cloud Control", and through the LCCC Linnaeus and ELLIIT Excellence Centers.

References

- [1] C. Pahl, "Containerization and the PaaS Cloud," *IEEE Cloud Comput.*, vol. 2, no. 3, pp. 24–31, May/June 2014.
- [2] J. P. Walters, V. Chaudhary, M. Cha, S. G. Jr., and S. Gallo, "A comparison of virtualization technologies for HPC," in *Proc. 22nd Int. Conf. Adv. Inf. Networking Appl. (AINA 2008)*. Okinawa, Japan: IEEE Computer Society, 25–28 Mar. 2008, pp. 861–868.
- [3] D. G. Feitelson, "Experimental computer science," *Commun. ACM*, vol. 50, no. 11, pp. 24–26, Nov. 2007.
- [4] Z. Li, L. O'Brien, and M. Kihl, "DoKnowMe: Towards a domain knowledge-driven methodology for performance evaluation," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 4, pp. 23–32, Mar. 2016.
- [5] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux J.*, vol. 239, pp. 76–91, Mar. 2014.
- [6] X. Xu, H. Yu, and X. Pei, "A novel resource scheduling approach in container based clouds," in *Proc. 17th IEEE Int. Conf. Comput. Sci. Eng. (CSE 2014)*. Chengdu, China: IEEE Computer Society, 19–21 Dec. 2014, pp. 257–264.
- [7] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, Sept. 2014.
- [8] M. G. Xavier, M. V. Neves, and C. A. F. D. Rose, "A performance comparison of container-based virtualization systems for MapReduce clusters," in *Proc. 22nd Euromicro Int. Conf. Parallel Distrib. Network-Based Process. (PDP 2014)*. Turin, Italy: IEEE Press, 12–14 Feb. 2014, pp. 299–306.
- [9] C. Anderson, "Docker," *IEEE Software*, vol. 32, no. 3, pp. 102–105, May/June 2015.
- [10] T. Banerjee, "Understanding the key differences between LXC and Docker," <https://www.flockport.com/lxc-vs-docker/>, Aug. 2014.
- [11] J. Che, C. Shi, Y. Yu, and W. Lin, "A synthetical performance evaluation of OpenVZ, Xen and KVM," in *Proc. 2010 IEEE Asia-Pacific Serv. Comput. Conf. (APSCC 2010)*. Hangzhou, China: IEEE Computer Society, 6–10 Dec. 2010, pp. 587–594.
- [12] D. Strauss, "Containers - not virtual machines - are the future Cloud," *Linux J.*, vol. 228, pp. 118–123, Apr. 2013.
- [13] T. Adufu, J. Choi, and Y. Kim, "Is container-based technology a winner for high performance scientific applications?" in *Proc. 17th Asia-Pacific Network Oper. Manage. Symp. (APNOMS 2015)*. Busan, Korea: IEEE Press, 19–21 Aug. 2015, pp. 507–510.
- [14] A. M. Joy, "Performance comparison between Linux containers and virtual machines," in *Proc. 2015 Int. Conf. Adv. Comput. Eng. Appl. (ICACEA 2015)*. Ghaziabad, India: IEEE Press, 14–15 Feb. 2015, pp. 507–510.
- [15] K.-T. Seo, H.-S. Hwang, I.-Y. Moon, O.-Y. Kwon, and B.-J. Kim, "Performance comparison analysis of Linux container and virtual machine for building Cloud," *Adv. Sci. Technol. Lett.*, vol. 66, pp. 105–111, Dec. 2014.
- [16] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *Proc. 2015 IEEE Int. Symp. Perform. Anal. Syst. Software (ISPASS 2015)*. Philadelphia, PA, USA: IEEE Press, 29–31 Mar. 2015, pp. 171–172.
- [17] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support PaaS," in *Proc. 2014 IEEE Int. Conf. Cloud Eng. (IC2E 2015)*. Boston, Massachusetts, USA: IEEE Computer Society, 10–14 Mar. 2014, pp. 610–614.
- [18] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Efficient virtual machine sizing for hosting containers as a service," in *Proc. 11th World Congr. Serv. (SERVICES 2015)*. New York, USA: IEEE Computer Society, 27 Jun.–2 Jul. 2015, pp. 31–38.
- [19] S. M. Blackburn, K. S. McKinley, R. Garner, C. Hoffmann, A. M. Khan, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. G. M. H. A. H. M. J. H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanovik, T. VanDrunen, D. von Dincklage, and B. Wiedermann, "Wake up and smell the coffee: Evaluation methodology for the 21st century," *Commun. ACM*, vol. 51, no. 8, pp. 83–89, Aug. 2008.
- [20] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production Cloud services," in *Proc. 11th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput. (CCGrid 2011)*. Newport Beach, CA, USA: IEEE Computer Society, 23–26 May 2011, pp. 104–113.
- [21] Z. Li, L. O'Brien, R. Cai, and H. Zhang, "Towards a taxonomy of performance evaluation of commercial Cloud services," in *Proc. 5th Int. Conf. Cloud Comput. (IEEE CLOUD 2012)*. Honolulu, Hawaii, USA: IEEE Computer Society, 24–29 Jun. 2012, pp. 344–351.
- [22] Z. Li, L. O'Brien, H. Zhang, and R. Cai, "A factor framework for experimental design for performance evaluation of commercial Cloud services," in *Proc. 4th Int. Conf. Cloud Comput. Technol. Sci. (CloudCom 2012)*. Taipei, Taiwan: IEEE Computer Society, 3–6 Dec. 2012, pp. 169–176.
- [23] StackOverflow, "What is the relationship between the docker host OS and the container base image OS?" <http://stackoverflow.com/questions/18786209/what-is-the-relationship-between-the-docker-host-os-and-the-container-base-image>, Sept. 2013.
- [24] Reddit, "Do I need to use an OS base image in my Dockerfile or will it default to the host OS?" https://www.reddit.com/r/docker/comments/2teskf/do_i_need_to_use_an_os_base_image_in_my/, Jan. 2015.
- [25] Z. Li, L. O'Brien, H. Zhang, and R. Cai, "On the conceptualization of performance evaluation of IaaS services," *IEEE Trans. Serv. Comput.*, vol. 7, no. 4, pp. 628–641, Oct.–Dec. 2014.