Arabian Journal for Science and Engineering (2021) 46:8585–8601
https://doi.org/10.1007/s13369-021-05553-3

RESEARCH ARTICLE-COMPUTER ENGINEERING AND COMPUTER SCIENCE

Virtualization in Cloud Computing: Moving from Hypervisor to Containerization—A Survey

Aditya Bhardwaj[1] · **C. Rama Krishna**[2]

Received: 24 April 2020 / Accepted: 9 March 2021 / Published online: 13 April 2021

Abstract
Containers emerged as a lightweight alternative to virtual machines that offer better microservice architecture support. They are widely used by organizations to deploy their increasingly diverse workloads derived from modern applications such as big data, IoT, and edge/fog computing in either proprietary clusters or private, public cloud data centers. With the growing interest in container-based virtualization technologies, the requirement to explore the deployment and orchestration of clusters of containers has become a central research problem. Although progress has been made to study containerization, systematic consolidation of the existing literature with a summative evaluation is still missing. To fill this gap, in this paper, we first taxonomically classify the existing research studies on the performance comparison between hypervisor and container technology and then analyze state-of-the-art for container cluster management orchestration systems, its performance monitoring tools, and finally future research trends. This results in a better understanding of container technology with attention to provide summative analysis in terms of (i) how much performance overhead is generated by a hypervisor compared to container-based virtualization, (ii) which container technology is suited for a cloud application deployment based on the type of benchmark executing, (iii) how to provide management of containers deployed in a cluster environment, (iv) container performance monitoring tools, and (v) finally emerging concerns for future research directions.

Keywords Cloud computing · Virtualization · Hypervisor · Containerization

1 Introduction

In the current technology era, cloud computing has completely reshaped the way end users can get access to computing and other IT resources. In the last few years, the cloud computing market has gained significant popularity. This is because it allows access to a large pool of resources in a pay-per-use, ubiquitous, cost-efficient, and virtualized manner. All these objectives can be achieved without worrying about maintenance overhead of the physical hardware [1–3]. Virtualization is the key technology that has revolutionized the working of cloud computing data centers. It enables the creation of multiple instances of VMs on the same physical infrastructure and thus improving resource utilization and increase in Return-on-Investment (ROI). Traditional hypervisor-based virtualization deployment creates significant performance overhead because each VM runs with a dedicated installed OS. This limitation hinders its application to High-Performance Computing (HPC) environment. CPU-intensive and data-intensive workloads are the commonly used HPC benchmark applications. Recently, a new paradigm of virtualization technology that has attracted considerable attention is containerization [4]. In containerization, applications can share their host OS kernel and contain only required binaries and libraries, making it lighter in size compared to a VM. To explore containerization, recent publications [5,6] have focused on the performance comparison between VMs and containers in a virtualization platform. From these research studies, it is inferred that compared to VM, a container provides an easy and convenient way to deploy cloud applications. However, there is a lack of studies to consolidate research on container technologies and identify gaps in terms of performance comparison with hypervisor, their cluster management, and monitoring tools perspectives [7].

✉ Aditya Bhardwaj
aditya.cse@nitttrchd.ac.in

[1] Department of CSE, KIET Group of Institutions, Delhi-NCR, Ghaziabad, India

[2] Department of CSE, NITTTR (Established by MHRD, Government of India), Chandigarh, India

Given the growing interest in container-based virtualization, there is a need to explore the challenges of performance comparison between hypervisor and container, orchestration systems, and monitoring tools. In our previous work [8], we investigated how to provide migration support for running containers. Further, this study is more comprehensive to consolidate the existing state-of-the-art research work based on the aforementioned challenging issues. But, it has less coverage from a network management perspective. We taxonomically classify and provide a summative evaluation of the existing literature studies on container technologies aiming to provide a better understanding in terms of (i) what is the overhead generated by hypervisor and container virtualization platforms, (ii) which container technology is suited for cloud deployment, (iii) the mechanism to provide orchestration support in a container, and (iv) finally classified container performance monitoring tools.

The authors believe that the systematic study carried out in this paper will provide a knowledge base for researchers, practitioners when they need to implement virtualization using container technology. From the existing literature, it is found that compared to VM, container provides a lightweight solution to deploy cloud applications. However, there is a need to address important challenging issues such as auto-scaling of containers, concern for security, failure management, dynamic resource allocation, and energy overhead. The remaining of this paper is organized as follows. Section 2 presents a background and motivation for virtualization technology. To map the existing literature systematically, the set of research questions formulated are listed in Sect. 3. Section 4 presents state-of-the-art for performance evaluation of hypervisor versus container. Further, a consolidated summary for the container orchestration system and its performance monitoring tools is presented in Sects. 5 and 6. Finally, research issues and challenges followed by concluding remarks are presented in Sects. 7 and 8.

1.1 Comparison Between the Existing State-of-the-Art and this Survey Paper

The existing literature reviews only addressed container-based virtualization from orchestration coverage perspective. It lacks in terms of performance comparison with hypervisor, their cluster management, and monitoring tools perspectives. However, to provide a global view, our study covers and analyzes literature from three different angles: (i) summative evaluation of performance comparison between hypervisor versus container (Sect. 4), (ii) characterization of key cluster management systems (Sect. 5), and (iii) consolidation of container performance monitoring tools (Sect. 6). A comparison of the existing literature studies and this survey paper is shown in Table 1.

Table 1 Comparison between the existing state-of-the-art reviews and this survey paper

L_p	T_p						
	RQ	RH	HC	CO	CM	NV	RC
[9]				✓			
[10]	✓	✓					
[7]		✓					
[11]		✓				✓	✓
[12]				✓			✓
[13]				✓			
[14]		✓				✓	✓
Our work	✓	✓	✓	✓	✓		✓

L_p, literature papers; T_p, topics covered; RQ, research questions; RH, reference architecture hypervisor versus container; HC, hypervisor versus container performance evaluation; CO, container orchestration; CM, container monitoring; NV, network virtualization; RC, research challenges

2 Background and Motivation

The existing hypervisor-based virtualization system deploys cloud applications using monolithic-service architecture. Therefore, the performance overheads created by the hypervisor system hinder its applicability in HPC environments. This has led to the emergence of new virtualization technology called containerization. This section first discusses the need for virtualization technology its evolution and finally motivation from the hypervisor to a container as next-generation virtualization technology.

2.1 Need of Virtualization in Cloud Computing Data Centers

In traditional computing architecture, a single physical machine is used to deploy limited applications only, which resulted in the underutilization of hardware resources. The concept of abstracting physical system resources into various virtual computing resources called virtualization was originated by IBM [15] and commercialized for the x86-computer system in 1990 [16]. Virtualization technique is considered as the backbone of cloud computing data centers because it enables deploying multiple virtual servers over the single physical system, thus improve resource utilization and increase the return-on-investment **(RQ1)**. Virtualization provides an abstraction over the physical resources that can be shared by cloud users.

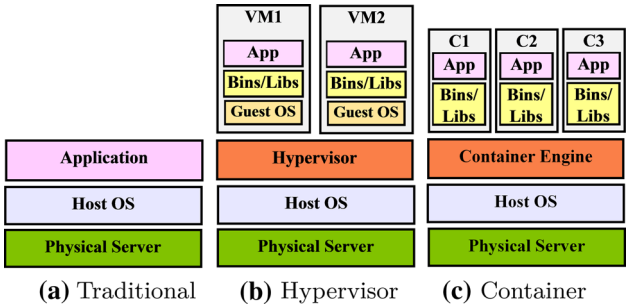

Fig. 1 Comparison between application deployment using traditional, hypervisor and container architecture

2.2 Evolution of Cloud Virtualization from Hypervisor to Containerization

In a cloud computing architecture, hypervisor-based virtualization has been widely used during the last decade to implement virtualization services through VM. Hypervisor-based deployment is ideal when applications on the cloud require resources and functionalities of a variety of operating systems. For example, these days industrial automation system needs both a real-time and general-purpose OS. KVM and Xen are the two most popular hypervisor technology [17]. However, one of the primary bottlenecks in hypervisor-enabled virtualization is that applications deployment using VM requires dedicated guest OS, binaries, library files, and thus considerably large VM image size [18]. Therefore, it has become a major concern for cloud service providers because application deployment using hypervisor-based virtualization results in system performance overhead. To provide solutions for these overheads, a new era of virtualization technique that has revolutionized the working of cloud data centers and provides a lightweight framework is called containerization **(RQ2)**. In container-based virtualization, sharing the host OS between the containers makes them very light **(RQ3)** as shown in Fig. 1. Compared to a VM, a container (e.g., Docker) can start in just 50 ms while a VM might take as long as 30 to 40 s to start, and restarting of the container does not require rebooting the OS [6]. Therefore, applications deployed on container-based virtualization improved resource utilization and it can be started faster compared to a VM with performance that closely resembles a bare-metal system (**RQ4**) [19].

However, one of the downsides of containerization technology is the lack of OS flexibility. This is because in a production environment each container must use the same OS as the host OS whereas hypervisor instances have more flexibility. Such solutions are challenged when there is a requirement of projects that are involved with their diverse OS and software demands. Thus, containers are suitable to deploy multiple instances of an application that runs on a single operating system infrastructure only.

Further, in container-based solutions, applications share a host OS, therefore there is an assumption that the security of the container-based virtualization approach is weaker than a hypervisor. However, for the Linux kernel the advanced features such as namespace and cgroups help to stabilize the security of the container. In this, the namespace feature is responsible to provide isolation among containers, and the control group (cgroup) provides management of resources like CPU, memory, disk I/O, network, etc.

LXC, Docker, and rkt are the three leading container technologies. LXC is a system container that allows running multiple applications on a single OS [20]. Alternative to LXC, Docker is an application container a project released in 2013 by dotCloud (now Docker Inc.) specifically designed to build a single app environment and does not provide a platform to run multiple applications [21]. However, security is the major flaw inherent in Docker containers because an attacker can easily get root-level access. To fix the security issue in the Docker container, in 2014, CoreOS launched rkt container with an important inbuilt feature of image signature verification [22].

3 Defining the Research Questions and Review Framework

The characterization of the review framework is shaped by formulating research questions from RQ1 to RQ16. These research questions and motivation formulated to derive a literature review are shown in Table 2. The main aim of defining the research questions is to map the existing work on container technologies systematically. We categorize the literature studies in terms of significant concerns such as container technology deployment, cluster orchestration systems, monitoring tools, and finally, emerging trends for future research directions. The layout of sections of this manuscript is structured as shown in Fig. 2.

4 State-of-the-Art for Hypervisor Versus Container Technologies Performance Evaluation

In this section, we review a performance evaluation-based studies of hypervisor and container virtualization technologies and their comparison with the bare-metal native platform. We first analyze studies based on performance overhead generated by the hypervisor and container-based virtualization, second identify which container is suited for a cloud application based on the category of workload. The studies based on the percentage difference between hyper-

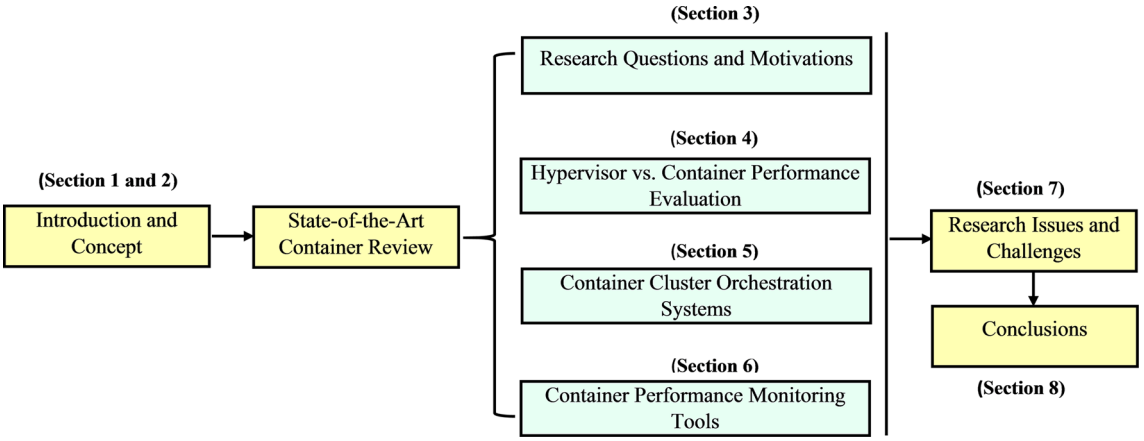

Fig. 2 Organization of literature review on container technologies

visor and container technologies w.r.t. native system are discussed in Section 4.1 and summarized in Tables 3, 4, and 5. For some set of virtualization test cases, the values are not available, which we calculated using our experimental setup [8] and highlighted by symbol *. Further, to visualize the data in graphical mode, we used bar graphs as shown in Figs. 3, 4, and 5.

4.1 Review on Performance Comparison of Hypervisor Versus Container Technologies

Felter et al. [23] have analyzed and compared the performance overhead of virtualization platform w.r.t. baseline system. To deploy the virtualization environment, the authors used KVM hypervisor and Docker container technology. Their benchmarks comprise CPU floating-point multiplication Intel Linpack [24], memory STREAM [25], netperf to measure bandwidth, network latency, and finally, Fio for block I/O [26]. Linpack benchmark measures performance of CPU system by generating linear equations and calculate the number of floating-point operations (FLOPS) a system can perform per second. In the STREAM benchmark, four operations, namely Copy, Scale, Add, and Triad are used to measure the performance of the memory system. Further, to carry out test cases, available cache memory is used to set the size of the 'STREAM array.' Netperf benchmark was used to measure the bandwidth and system latency. For the execution of the Linpack benchmark, compared to the non-virtualization base system, the performance overhead generated by the Docker container is just 2.42%, but KVM performed worst with a difference of 51.78%. This is because in the KVM hypervisor each time for an I/O operation, there has to be context switching. Therefore, hypervisor-based virtualization does not fulfill the requirement of a HPC environment **(RQ5)**. Furthermore, it is also inferred that compared to virtual machines, Docker container performs similar to the native system and produces negligible overhead. Therefore, adopting a container as virtualization technology would provide benefit to the cloud providers.

Morabito et al. [27] also explored the performance comparison study between hypervisor versus container platform. But, as compared to [23], the authors in this study considered Docker and LXC as container technology. Docker is an application container specifically designed to run a single application solution. However, LXC is a system container that allows the user to run multiple isolated images on a single host. From the workload categories, the authors considered Y-cruncher [28], NBENCH as CPU-intensive benchmarks, Bonnie++ [29], Netperf, and disk write (dd test) for the network-intensive workload. From their experimental results, it is found that the amount of overhead introduced by both Docker and LXC container technologies is relatively small compared to the KVM hypervisor. Therefore, cloud providers can be benefited from virtualization deployment using container technology.

Furthermore, Kozhirbayev and Sinnott [30], conducted a performance comparison between Docker and LXC container virtualization technologies w.r.t. native platform. They explored CPU (Y-cruncher, Linpack), memory (Geekbench STREAM), Disk (Bonnie++, Sysbench), and network I/O benchmarks. In the CPU category, Y-cruncher is used to compute the constant value of Pi, Linpack test system performance by solving a set of linear equations. In memory benchmark, Geekbench tests RAM performance by executing integer and floating-point applications [31], STREAM tool measures memory I/O performance using its basic set of operations. Furthermore, to assess the disk I/O performance, Bonnie++ with a 4 GB dataset was used to test sequential/random, read, write operations and Sysbench with 35 GB test file for the cross-platform environment. Finally, Netperf and Iperf with TCP, UDP protocols were used to test network throughput between a client and server. Their results indicate

Table 2 Defining the research questions on cloud container technology

No.	Research Questions (RQ)	Motivation
1.	**Container Technology Deployment** How to maximize resource utilization in cloud data centers virtualization environment?	The aim of these research questions is to analyze and identify performance overhead caused by hypervisor-based virtualization and how container-based solution can address these issues.
2.	Why lightweight virtualization is required in the cloud?	This information can help the researchers, practitioner to investigate hypervisor overhead and make a selection for container technology according to the running application requirement.
3.	How to deploy container-based virtualization environment?	
4.	How container can be a game changer for data center virtualization technology?	
5.	Does the existing hypervisor fulfill requirement of High-Performance Computing (HPC) environment?	
6.	What is the overhead generated by hypervisor compared to container-based virtualization?	
7.	Which container is suited for cloud application deployment based on the benchmark category?	
8.	**Container Orchestration Systems** What are the challenges when containers are deployed in clustered environment?	With increase in demand of cloud services, it becomes essential to define and develop clustering technique which can effectively manage a large number of container deployment.
9.	How to facilitate management and orchestration for clusters of containers?	These research questions will help in selection of container clustering mechanism.
10.	What are container orchestration systems available?	
11.	How containers are connected in clustered environment?	
12.	How orchestration system can play a key role for edge and IoT computing platform?	
13.	**Container Monitoring Tools** What are the key performance metrics in container technology?	Finding the root causes for anomalous events requires better resource monitoring.
14.	What are the tools available for monitoring of container performance?	It is interesting to explore monitoring tools for container performance metrics.
15.	**Container Research Challenges** What are the major concerns in adopting container technology?	There is a need to consolidate the existing study on container technology to identify research gaps and extract pointers for the future direction.
16.	What are the key challenges in deployment of container technology?	

that for the test case of CPU and memory benchmark KVM hypervisor performed worst with a performance overhead of 33.36% and 15.67%, and Docker, LXC container technologies produce negligible overheads that range from 0.46% to 3.20%. However, for the network-intensive application, these container technologies also provide a small overhead due to the requirement of additional cycles to complete the I/O operation. Therefore, it is inferred that Docker, LXC containers have more advantages for benchmarks with lower I/O operations. As a consequence, application deployment using container technology can provide better results and performance close to a native system compared to launching virtual machines using the hypervisor.

Xiao-Lan et al. [32] conducted a performance comparison of container technologies, namely, Docker and rkt [22] w.r.t. native platform. Their experimental tests were conducted using CPU and disk I/O benchmarks. In the CPU-intensive benchmark, they considered Y-cruncher, and in the disk I/O category, Bonnie++ benchmark was used. Y-cruncher is used to compute the value of Pi, and it is measured in terms of computation, execution time. Bonnie++ is an open-source benchmark used for block input/output tests. In the first category, the performance measured using the Y-cruncher benchmark indicates that both Docker and rkt performs close to the native system, however, for disk read operation per-

formance of rkt container is better than Docker. This is because rkt container provides faster access to disk data using *overlay* file system. From this study, it is inferred that the performance of container-based virtualization is close to the native system. Therefore, in a cloud data center, applications can be deployed using container technology as compared to hypervisor-based virtualization.

Marting et al. [6], extended the study of [32] with CoreOS rkt as the emerging container virtualization technology. This container technology focused on ease of application development. To carry out their study, the authors considered Linpack (compute-intensive) and Graph500 (data-intensive) benchmarks [33] with varying edge factor from 16 to 36. Linpack tests system performance by solving a set of linear equations and measured in terms of GFLOPS (Billions of floating-point operations per second). In Graph500, a large-scale graph with data generation and Breadth-First Search (BFS) was executed. Data traceability is the criteria used for evaluation of the Graph500 benchmark and it is measured in Traversed Edge Per Second (TEPS). To evaluate the results, the authors considered LXC, Docker, and rkt as the representative of container virtualization technologies. From their experimental results, it is found that Docker causes performance degradation for CPU and data-intensive benchmarks. This is because the Docker container causes context switching overhead. Further, for data-intensive (Graph500), and compute-intensive (Linpack) benchmark the performance measured value is 54.9×10^7 TEPS, 96.1 GFLOPS when executed on LXC container, and 51.8×10^7 TEPS, 136.6 GFLOPS on rkt container. For performance evaluation, the higher value of TEPS and GFLOPS indicates better results. Thus, it is inferred that the LXC container shows better performance for the data-intensive environment (Graph500), and the rkt container gives promising results in the compute-intensive (Linpack) benchmark. This is because the absence of the daemon process in the rkt container engine resolves compatibility issues during the runtime environment and increases the startup time.

In Google Cloud Platform [34], Google cloud service providers encourage container utilization for the application deployment in Platform-as-a-Service (PaaS) architecture. In PaaS, the client outsources the development tools framework to execute its application. In the Google data center, everything runs on the container from Gmail to YouTube search. Containers offer a lightweight solution with the same isolation feature as provided by the virtual machine. A similar observation was made in [35]. However, usage of container for PaaS is still in its development phase, and there is a need to explore issues of container deployment in a cluster and migration environment.

Yadav et al. [36] explored the performance comparison between VM and Docker container technology in terms of OS support, boot time, server density, access to hardware, resource utilization, and memory requirement. Virtual machine-based virtualization is deployed using dedicated hardware OS files while the container contains necessary library packages and dependencies associated with the applications. From a booting time perspective, a Docker container can boot faster as compare to a VM because virtualization deployment using a container is less resource-centric. While working for the HPC environment applications, Docker is a cost-effective solution because a large number of applications can be accommodated on the same server as compared to VMs. Compared to VMs containers are less resource-centric, provide lightweight virtualization architecture, and less performance overhead. Thus, containers are advantageous over virtual machines. Due to these features, a container-based solution results in a better solution compared to hypervisor-based virtualization deployment. However, security is a challenging issue in container-based virtualization. This is because the host kernel is shared so that a single vulnerable point can lead to hacking of the entire server. Thus, there is a need to explore security issues in container-based virtualization.

Luo et al. [37] also explored the study of a container and VM-based application deployment. However, compared to [36], the authors in this study discussed the issue of the fog computing environment. To solve the problem regarding resource development, they proposed a container-based multi-cloud to multi-fog architecture intending to reduce the response time of requests from fog nodes and minimize the service delay. In their study, benchmarks such as Linpack and MySQL were executed on the terminal device to evaluate the performance of VM and container under concurrency condition in terms of per-request execution time and the number of transactions completed per second. Tests were carried out when the number of containers are run on VM, or the number of VMs are running on KVM. Experimental results show that containers running on VM enhance the isolation and outperform the VMs running on KVM with a reduction of 26.95% for execution time and 22.32% increase in more transactions processed per second. However, further studies are required to discuss these virtualization technologies for fog computing architecture in the scenario of a node failure, clustering framework, and migration environment.

4.2 Benchmarks-Based Graphical Visualization of Hypervisor Versus Container Technologies

In this subsection, we consider graphical representation to visualize and ease the presentation for our main findings that are related to the state-of-the-art hypervisor and container-based performance comparison studies. These studies, according to the percentage of performance difference compared to the native baseline system for CPU,

Table 3 Performance comparison for CPU-intensive benchmarks

Benchmark	Purpose	% of performance overhead compared to non-virtualized system				Findings
		KVM	Docker	LXC	rkt	
NBENCH [27]	It is a test to measure CPU performance in terms of integer, floating, and memory index in a single-threaded environment.	27.53%	0.36%	0.47%	0.23*%	From this test, it is found that in single-threaded benchmark KVM performed worst and container gives near native performance.
Y-cruncher [30]	It is used as a stress testing tool for CPU and calculate value of Pi in multi-threaded environment.	33.36%	1.12%	3.20%	0.55%	In multi-threaded benchmark, KVM performed worst and container gives near native performance. In container category, rkt outperforms Docker and LXC container.
Linpack [23]	This benchmark is based on linear algebra and solves linear equations using lower upper decomposition.	51.78 %	2.42%	3.60%	1.73%	For linear algebra benchmark, compared to non-virtualized system, KVM performed worst and in container-based virtualization category rkt outperforms Docker and LXC container.
Pbzip2 [30]	It is a data compression benchmark where a file is compressed using p number of threads.	55.33*%	8.02%	8.75%	7.49*%	From this benchmark test, it is found that rkt performed slightly better than LXC and Docker.

*Means values calculated using our experimental setup

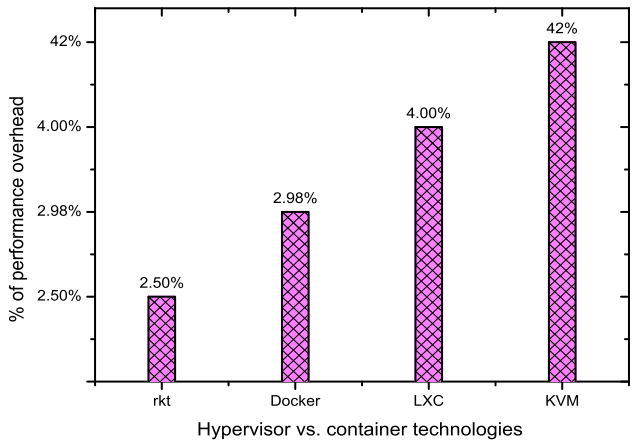

Fig. 3 Graphical evaluation of hypervisor versus container technologies for CPU-intensive benchmarks

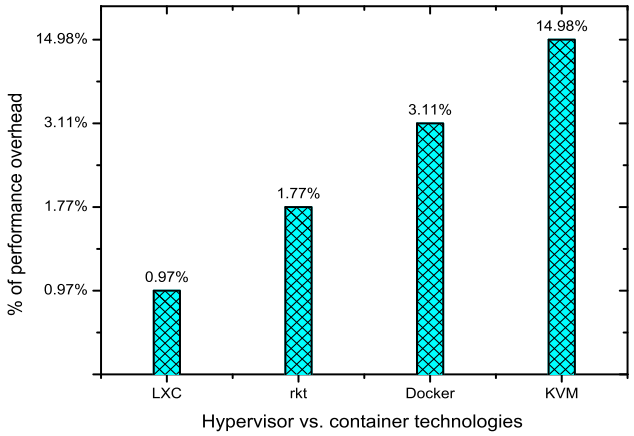

Fig. 4 Graphical evaluation of hypervisor versus container technologies for memory-intensive benchmarks

memory, network, and disk-intensive benchmarks, are summarized in Figs. 3, 4, and 5.

4.2.1 CPU-Intensive Benchmarks

In this category, NBENCH, Y-cruncher, Linpack, and Pbzip2 are widely used benchmarks. NBENCH and Y-cruncher benchmarks are used to test CPU performance when it is processed in a single and multi-threaded environment. Linpack measures the performance of the CPU system using linear equations. Pbizip2 is used to evaluate the processing of the CPU during the operation of compressing a file. For the CPU-intensive benchmark test case, the performance results of KVM hypervisor and container-based virtualization technologies w.r.t. native system are shown in Table 3. From the analysis of the CPU-intensive benchmark, it is inferred that compared to the non-virtualized system, KVM performed worst with a performance difference of 42% **(RQ6)** and container-based virtualization performs similar to the bare-metal (non-virtualized) system. This is because, in KVM, VM execution consumes CPU resources to tune itself to the running state over the architecture upon which it is running. In the container category, the percentage of performance difference by rkt, Docker, and LXC is 2.50%, 2.98%, 4.00%, respectively. In container technology, rkt outperforms Docker and LXC. This is because unlike Docker and LXC, rkt container has no centralized daemon process and thus resolves compatibility issues with Linux *init* system. Therefore, applications that are CPU-intensive can be deployed using rkt container technology **(RQ7)**.

4.2.2 Memory Intensive Benchmarks

In this category, STREAM, Graph500, FEniCS, and Geekbench are the widely used memory-intensive benchmarks. STREAM is used to measure memory throughput using its basic set of operations. Graph500 is used to calculate BFS of a large undirected graph, FEniCS for distributed memory, and Geekbench executes integer and floating-point workloads.

In memory-intensive benchmarks, the percentage of performance difference introduced by hypervisor and container technologies is KVM 14.98%, LXC 0.97%, rkt 1.77%, Docker 3.11%. Therefore, in memory-intensive benchmarks compared to hypervisor-based virtualization, the container provides low overhead. In the container category, LXC outperforms Docker and rkt. This is because LXC containers are designed to run full system container images; thus, they can provide sufficient memory to execute a memory-intensive application. Therefore, it can be inferred that applications that are memory intensive can be deployed using LXC container technology **(RQ7)**.

4.2.3 Network and Disk-Intensive Benchmarks

The benchmarks from this category are used to test the performance of the network, disk input/output operations. Sysbench, dd test, Bonnie++, and Netperf are the widely used network and disk-intensive benchmarks. From the results obtained, it is found that the percentage of performance difference introduced by KVM hypervisor is 48.29%, and container technologies are rkt 6.70%, Docker 7.86%, LXC 17.81%.

Therefore, for the network, disk-intensive benchmarks also hypervisor-based virtualization introduces significant overhead, and container-based virtualization provides low overhead. In the container category, rkt container performs better than LXC and Docker because it uses a secure open-source file system called overlay, which supports faster access of disk data by the user [39].

Thus, it can be inferred that rkt container is suited for network and disk-intensive benchmarks (**RQ7**.)

Table 4 Performance comparison for memory-intensive benchmarks

Benchmark	Purpose	% of performance overhead compared to non-virtualized system				Findings
		KVM	Docker	LXC	rkt	
STREAM [23]	It is used to measure memory input/output performance using Copy, Scale, Add, and Triad operation.	13.03%	2.54%	1.47%	1.91*%	In this test-case, KVM introduced significant amount of overhead and performance of container virtualization is nearly same as the native platform. Also, in container category, LXC produces slightly better results than Docker and rkt.
Graph500 [6]	It is an analytic data-intensive benchmark which executes graph data structure to obtain BFS in data generation and construction phase.	16.49%	7.07%	1.08%	3.62%	In data-intensive memory benchmark, the LXC container performed close to bare-metal system and shows better compared to Docker and rkt.
FEniCS [38]	It is used for distributed memory parallelism and solves Poisson equation using the conjugate gradient method.	14.75%	2.15%	0.89%	0.97%	From this memory test, it is inferred that compared to LXC, Docker, and rkt containers, KVM introduced significant overhead.
Geekbench [30]	This benchmark tests performance of memory system by executing workloads using multiple indexes such as integer and floating point.	15.67%	0.68%	0.46%	0.59*%	It is found that for single and multi-core architecture, KVM performs worst and LXC shows better performance compared to Docker, and rkt container.

*Means values calculated using our experimental setup

Table 5 Performance comparison for network and disk-intensive benchmarks

Benchmark	Purpose	% of performance overhead compared to non-virtualized system				Findings
		KVM	Docker	LXC	rkt	
Sysbench [30]	It is used to measure disk I/O throughput based on read and write operations from a specified file.	39.78%	0.61%	4.35%	0.45*%	KVM has higher latency because of overhead of entering and exiting the VM.
dd test [27]	This benchmark is used to test read and write capacity of the disk system.	59.18%	7.37%	24.59%	5.52*%	KVM introduced large overhead and rkt performs better than LXC and Docker because it uses *overlay* file system which supports faster access of disk data by the user.
Bonnie++ [27,32]	This benchmark is used to test read/write capacity of system's disk.	50.26%	14.68%	20.53%	13.63%	In this test case also, performance overhead of hypervisor virtualization is high compared to a containerization.
Netperf [30]	It is used to measure network performance using TCP and UDP data transfer between a client and server.	43.96%	8.81%	21.79%	7.20*%	For TCP and UDP stream test cases, KVM introduced largest overhead and rkt performs better than LXC and Docker.

*Means values calculated using our experimental setup.

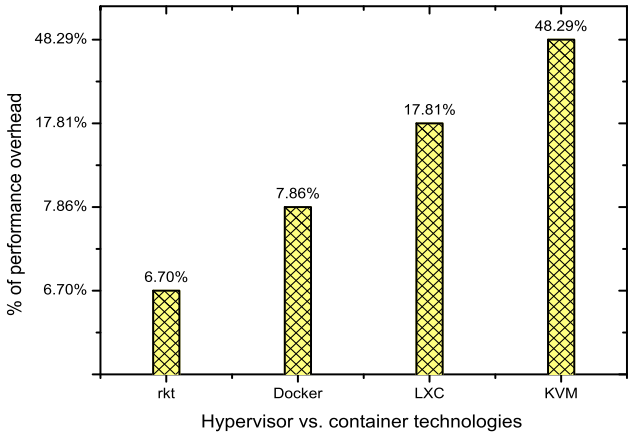

Fig. 5 Graphical evaluation of hypervisor versus container technologies for network and disk-intensive benchmarks

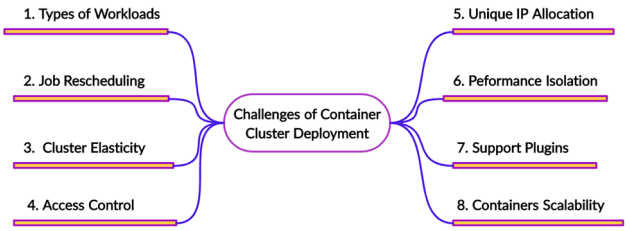

Fig. 6 Challenges of container cluster deployment

5 State-of-the-Art for Container Cluster Orchestration Systems

In recent years, container-based virtualization has been widely used by the cloud service providers to deploy applications such as web servers, big data, IoT, and edge/fog computing environment. This, in turn, has led to challenges of executing heterogeneous types of workloads, job rescheduling, autoscaling of cluster nodes, resource management, allocating unique IP address per container, support of third-party plug-in, and finally functionality to scale for large cluster infrastructure **(RQ8)**. Figure 6 shows the pictorial representation of these challenges. To meet these issues, the container orchestration platform has gained significant attention for the management of cloud applications deployed over a cluster of containers. A container orchestration system can manage hundreds or thousands of container applications running across a large number of systems cluster **(RQ9)**. The choice between a good container orchestration system depends upon the functionality supported by it. Therefore, in Sects. 5.1 and 5.2, we segregated the study based on two widely adopted container orchestration systems, namely Docker Swarm [40,41] and Kubernetes [42] **(RQ10)**.

5.1 Docker Swarm

In the landscape of container cluster management solutions, the leading open-source solutions are Docker Swarm and Kubernetes. Docker Swarm was specifically designed for the management of a cluster of Docker containers only. Docker Swarm is a container orchestration tool to manage and control multiple Docker containers as a single service. In the Docker Swarm orchestration platform, one machine is created as a Swarm manager and others as workers. Therefore, all the control and management operations of Docker containers can be done using the Swarm manager.

5.2 Kubernetes

Kubernetes, an orchestration system for Docker, LXC, and rkt containers, is capable of automating deployment, scaling, and load balancing of containers. Kubernetes was originally developed by Google and later acquired by Cloud Native Computing Foundation (CNCF) [43,44].

From the implementation perspective, a Kubernetes cluster consists of the installation of master and worker nodes. In the master node, the four main components are the API server, scheduler, control manager, and ETCD storage system. The API server is responsible for making communication with the worker node. Scheduler and control manager acts as the Kubernetes control plane to place the container on cluster based on the availability of resources **(RQ11)**. ETCD is a database that uses a key-value pair format to store container cluster information. A worker node is responsible for hosting containers application using a basic building block called a pod. This node consists of two main components, namely kubelet and kube-proxy. A kubelet communicate with the API server to gather basic information such as CPU, RAM, and network usage of running container pod application. The second component of the worker node, i.e., kube-proxy, also communicates with the API server to forward TCP/UDP traffic using configured IP tables. In a production environment, Kubernetes can support clusters of 300,000 containers nodes [42]. Thus, it can fulfill the requirement of cloud industries.

The studies based on Docker Swarm and Kubernetes container orchestration system are summarized in Table 6. It is found that compared to Kubernetes, Docker Swarm was specially designed for scheduling of Docker containers only and lacks the applicability to LXC and rkt container technologies. The system architecture implementation of Docker Swarm consists of a Swarm manager. It is a set of nodes with at least one master node and several worker nodes that can be virtual or physical machines. Docker Swarm is capable of scheduling a cluster of 30,000 (thirty thousand) containers [40]. However, compared to Kubernetes, it lacks the key functionality of fault tolerance (capability to handle failures of nodes) and resource management of containers to meet the spike

Table 6 Comparison of Docker Swarm and Kubernetes container orchestration systems

S. No.	Parameter	Docker Swarm	Kubernetes
1	Open-source	Yes	Yes
2	Target containers	Docker	LXC, Docker, rkt
3	Workload supported	Long-running job	All types of jobs
4	Job rescheduling	No	Yes
5	Cluster elasticity	Manual	Auto-scaling limited to homogeneous cluster
6	Access control	No	Yes
7	Network IP isolation	Yes	Yes
8	Performance isolation	No	Yes
9	Third party plugin support	Yes	Yes
10	Number of containers supported	30,000	300,000

Table 7 State-of-the-art container performance monitoring tools

Parameter	Docker-stats	cAdvisor	Prometheus	Sensu	Sysdig
Developed by	Docker	Google	SoundCloud	Sensu	Sysdig
License	Open-source	Open-source	Open-source	Open-source, commercial	Open-source, commercial
Target containers	Docker only	Docker only	Docker only	Docker, LXC, and rkt	Docker, LXC, and rkt
Performance metrics measured	CPU, memory, and network I/O usage	CPU and memory usage	CPU and memory usage	CPU, memory, and network I/O usage	CPU, memory, and network I/O usage
OS supported	Linux	Linux	Linux, Windows	Linux, Windows	Linux, Windows
Type of Interface	CLI	GUI	GUI	CLI/GUI	CLI/GUI
Alert	No	No	Yes	Yes	Yes
Service Kubernetes support	No	Yes	Yes	Yes	Yes
Visualization tools support	No	Grafana	Node Exporter and Grafana	Grafana	Sysdig Monitor
Histogram of resource usage	No	Yes	Yes	No	Yes
Extensibility of API	Yes	Yes	Yes	Yes	Yes
Ease of Install (score out of 5)	5	4	3	2	4

Note: 5 means easy to install

during application traffic. In Kubernetes, the functionality of fault tolerance is supported by using the replication control approach, and resource management is provided based on the utilization of RAM and CPU metrics. As a result, Kubernetes is a leading container management tool in the market and can play a key role in terms of resource management for application deployment using cloud, edge, and IoT computing platform **(RQ12)**.

6 State-of-the-Art for Container Performance Monitoring Tools

As containers run in their own namespaces, traditional Linux monitoring tools such as *top*, *ps*, *tcpdump*, and *lsof* from the host system do not help to monitor the activity happening within a container or between containers [45]. Therefore, the characterization of container performance monitoring tools is also one of the important, challenging issues that need to be addressed. Also, the choice of a good container monitoring tool by cloud industries acts as a key performance indicator and increases the chances of conformance to the Service Level Agreement (SLA). However, the existing literature lacks such studies. To address this issue, in this section, we presented a summative study on container monitoring tools that can help to get full visibility inside every container. To monitor resource usages of Docker container, the simple solution is Docker stats command [46]. It is an open-source default API available in the Docker daemon and can provide

resource usage statistics of a running Docker container in terms of CPU, memory, and network I/O **(RQ13)**. However, Docker stats is based on Command Line Interface (CLI) and is limited to a small scale only.

Further, to analyze Docker container metrics in the graphical web interface, in 2014, Google launched a cAdvisor that monitors CPU and memory usage metrics [47]. However, cAdvisor does not provide the facility of sending an alert. As an improvement to cAdvisor, in 2016, SoundCloud launched the Prometheus tool that provides the facility of alert based on applied rules [48]. Prometheus was widely adopted because it is based on key-value pairs that can be used easily for multi-dimensional data (containing lots of different labels per metric). However, the applicability of Docker stats, cAdvisor, and Prometheus tools is limited to monitoring of Docker container only and lacks the consideration for LXC and rkt container technologies.

To address the limited applicability of Docker stats, cAdvisor, and Prometheus tools, Sensu [49] and Sysdig [50] are the container monitoring tools that provide support for Docker, LXC, and rkt all the three containers technologies. Sensu is a self-hosted and centralized metrics service. However, the deployment of Sensu depends upon various complementary services such as Redis, RabitMQ, Sensu API, Uchiwa, and Sensu Core; thus, it leads to a delay in service. Compared to Sensu, Sysdig is the most widely adopted container monitoring tool. This is because Sysdig is easy to deploy and just needs the installation of required packages on a single host system only. Sysdig also provides alerting criteria with great visualization tools for detailed monitored metrics. Thus, if a user wants to monitor Docker, LXC, or rkt container, then the Sysdig tool can be a good choice **(RQ14)**. The findings related to container monitoring tools are summarized in Table 7.

7 Open Research Issues and Challenges

There is considerable interest in the adaptability of container-based virtualization. The open research issues and challenges in the hypervisor and container-based virtualization that need to be addressed are described in this section. From the existing literature, it is found that the issue of performance overhead problem in hypervisor-based virtualization can be addressed by adopting container technologies. Given the strong increase in the adaptability of container-based virtualization, the challenges of auto-scaling, container security, failure management, resource allocation, fault tolerance, energy consumption, affinity-based placements, co-located containers interference, network communication overhead, and how to deploy NFS services are at significant concerns **(RQ15)**. A detailed description of these research challenges is described in the following:

1. **Auto-scaling of container:** Through orchestration support, containerization facilitates the application deployment from a single host to a cluster of containers. To satisfy the stated objectives in the Service Level Agreement (SLA) and maintain high availability of container services, there is a need to automate container orchestration mechanism by using machine learning techniques **(RQ16)** [51,52]. This automated system would facilitate adding or deleting a new container in cluster architecture based on the current utilization of resources. This will improve the performance of cluster architecture along with an improvement in the availability of cloud services.
2. **Security concerns in container-based virtualization:** Container-based virtualization emerged as a lightweight alternative to VMs. This greatly reduces start-up time and required resources for each application deployment. However, one of the primary adoption barriers for container widespread deployment is the security issues that are discussed in the following:
   (i) Security issues during container migration: In cloud computing, container migration has become an important technique because of its contribution to the consolidation of services. However, it can be vulnerable to various kinds of attacks such as eavesdropping, man-in-the-middle, and denial-of-service (DoS). During container migration, the issues that need to be addressed while preserving its security are: defining access control policy, preserving communication connectivity, and periodic update of the log file for source and destination server activities [53].
   (ii) Blockchain for image verification: Existing studies [54,55] showed that Docker images contain high-risk vulnerabilities that range from 30% to 90%. Thus, unverified container images can lead to an authentication issue. Therefore, it is interesting to explore blockchain decentralized architecture for container image verification [56].
   (iii) Container vulnerability scanning tools: There is a need to explore the usability of container image scanner vulnerable tools in terms of performance, automation, and integration with orchestration systems. This would provide great benefits for developers and practitioners to examine the risks associated with a container image [57].
3. **Failure management:** The common causes of container failures are (i) misconfiguration of the container, (ii) over-commitment of resources, (iii) attempting to load workloads on containers that are not suitable to execute. To provide a solution for failure management of container, there is a need to explore techniques for log analysis and container failure prediction mechanism [12].
4. **Dynamic resource allocation:** In a cloud container data center, whenever an application requires the resources in terms of CPU and memory, then they are statically allocated by the orchestration system. However, with

the increase in diverse workloads such as big data, IoT, scientific application, edge computing [58,59], and web services, the static-manual allocation approach leads to underutilization of resources and an increase in the operational cost of the organization. Therefore, there is a need for a dynamic resource allocation scheme that can work based on the heterogeneity of workload characteristics [60,61].

5. **Need of fault tolerance multi-tenancy mechanism:** To deploy cloud applications compared to virtual machines, containers provide a lightweight approach as they share the host OS kernel. However, sharing kernel may result in an inevitable side-effect because the crash of the host's OS due to the malicious container may also cause other applications' failure. So, as future research work, it is interesting to explore how to facilitate a fault tolerance isolation mechanism. [12].
6. **Energy consumption:** Another important research direction for container-based virtualization is to explore from the energy consumption perspective when it is in idle, load testing, and unused state. A VM or container is considered as unused when they are no longer in use at all usually from the last 30 days [62]. The major problem in unused VMs and containers is that they result in a substantial increase in resource usage cost. Thus, there is a need to explore an approach that can balance the energy consumption of container virtualization.
7. **Issues in affinity-based placements:** In the cloud data center, container placement has become one of the key techniques to improve resource utilization. The basic principle of container placement is to allocate the maximum number of containers on both physical and VM's nodes while satisfying constraints of less energy consumption, decreasing the number of container migration, and assurance of SLA [63].
   For container placement, the intuitive and effective solution is to co-locate the containers which have an affinity to others when deployed on the same machine. However, in affinity-based container placement, the challenges that directly affect the overall performance and thus needs to be taken into account are i) efficient scheduling of concurrent containers request [64], ii) network conditions to identify the location of data, and iii) determining at which node migrating container should be placed [65].
8. **Interference on containers:** Containers promise bare-metal performance, but it may suffer from performance interference in a multi-tenant scenario [66]. In a container, co-located applications can cause performance interference, and the degree of interference is higher for certain types of workloads such as memory and disk I/O-intensive benchmarks [67]. As future work, studies are required in terms of mechanisms to provide control access to host OS resources and enabling the scheduler that can perform actions depending on the interference characteristics of running applications.
9. **Network communication model:** Virtual networks allow the virtual machine to communicate with the rest of your network, the host machine, and other virtual machines. In VM networking, bridge, network address translation (NAT), and routed modes are the suitable solutions. Container networking is similar to VM networking with two major variations. One is that containers are ephemeral; second is as compared to VM, container support maximum scalability thus there is a need for large address space [68].
   The choice of an inter-container communication model depends upon balancing the trade-offs between performance, security, and isolation. In container networking, a bridge for a single network and overlay (used in Docker Swarm and Kubernetes) for multiple networks are the suitable solutions [69]. In a bridge network, security is provided by using isolated namespaces and in an overlay network security is ensured using packet encapsulation and decapsulation process. Further, in container networking, there is a need to address important challenging issues such as optimization of the existing container networks and consideration of different workload characteristics [70].
10. **Challenges to deploy NFV services:** Network Functions Virtualization (NFV) is a way to virtualize network services, such as routers, firewalls, and load balancers, that have traditionally been run on proprietary hardware [71]. VM deployments were not enough considering the scale and demand for agility in service launches and low latency requirements. In NFV infrastructure as an alternative to VM, container-based virtualization brings a significant amount of benefits like reducing resource consumption (CPU, memory, etc.), rapid deployment with minimal run-time requirements, autoscaling, failure management, and better service agility [72].
    For future research, it is interesting to explore the applicability of container-based architecture for NFV into other scenarios such as 5G, IoT, and big data. Also, given the dynamic requirements of NFV, there is a need for resource allocation policies that can find solutions using container technology.

8 Conclusions

In this paper, the literature on the hypervisor and container-based virtualization is consolidated by formulating a research question framework based on key concerns namely application deployment using hypervisor versus container virtualization platform, container orchestration systems, perfor-

mance monitoring tools, and finally future research challenges.

From the summative evaluation of the existing state-of-the-art studies on application deployment using hypervisor versus container virtualization, it is found that compared to bare-metal (non-virtualized) system, KVM performed worst with a performance difference of 42% for CPU-intensive, 14.98% for memory-intensive, and 48.29% for network disk-intensive benchmarks. Thus, the overhead associated with the hypervisor-based virtualization platform hinders its usage for High-Performance Computing (HPC) environment. However, a container-based virtualization platform gives almost the same performance as a bare-metal system. Regarding the choice from container category, it is found that the rkt container is well suited for CPU-intensive and network disk-intensive workloads, and for memory-intensive benchmarks, the LXC container outperforms other container technologies.

Further, for the management of a cluster of containers it is found that compared to Docker Swarm, Kubernetes is the widely used container orchestration management tool that provides important functionality such as fault tolerance, scalability, access control, and efficient resource utilization.

The selection of a suitable container monitoring tool is also one of the important challenging issues. The findings related to container monitoring tools demonstrate that Sysdig is the most widely adopted container monitoring tool. This is because compared to other container monitoring tools, Sysdig provides several capabilities such as easy to install, alert service, visualization support, and histogram of resource usage.

To conclude, next-generation cloud data centers are moving from the hypervisor to containerization a lightweight application deployment platform, and that requires strong contribution from the research community focuses on open research challenges of (i) auto-scaling, (ii) security, (iii) failure management, (iv) dynamic resource allocation, (v) need of fault tolerance system, (vi) issues of energy consumption, (vii) efficient scheduling, (viii) co-located containers interference, (ix) network container communication overhead, and x) deployment for NFV-based services.

References

1. Lai, J.; Tian, J.; Zhang, K.; Yang, Z.; Jiang, D.: Network emulation as a service (NEAAS): towards a cloud-based network emulation platform. In: Mobile Networks and Applications, pp. 1–15 (2020)
2. Younas, M.; Jawawi, D.; Ghani, I.; Shah, M.A.; Khurshid, M.M.; Madni, S.H.H.: Framework for agile development using cloud computing: a survey. Arab. J. Sci. Eng. **44**(11), 8989–9005 (2019)
3. Al-Ruithe, M.; Benkhelifa, E.; Hameed, K.: A systematic literature review of data governance and cloud data governance. Pers. Ubiquit. Comput. **23**(5–6), 839–859 (2019)
4. Kristiani, E.; Yang, C.-T.; Wang, Y.-T.; Huang, C.-Y.; Ko, P.-C.: Container-based virtualization for real-time data streaming processing on the edge computing architecture. In: Proceedings of International Conference on Wireless Internet, pp. 203–211 (2018)
5. Chae, M.; Lee, H.; Lee, K.: A performance comparison of linux containers and virtual machines using docker and KVM. In: Cluster Computing, pp. 1–11 (2017)
6. Martin, J.P.; Kandasamy, A.; Chandrasekaran, K.: Exploring the support for high performance applications in the container runtime environment. Human-centric Comput. Inf. Sci. **8**(1), 1–15 (2018)
7. Bachiega, N.G.; Souza, P.S.; Bruschi, S.M.; de Souza, S.d.R.: Container-based performance evaluation: a survey and challenges. In: Proceedings of IEEE International Conference on Cloud Engineering, pp. 398–403 (2018)
8. Bhardwaj, A.; Krishna, C.R.: A container-based technique to improve virtual machine migration in cloud computing. In: IETE Journal of Research, pp. 1–16 (2019)
9. Heidari, P.; Lemieux, Y.; Shami, A.: Qos assurance with light virtualization-a survey. In: Proceedings of IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 558–563 (2016)
10. Pahl, C.; Brogi, A.; Soldani, J.; Jamshidi, P.: Cloud container technologies: a state-of-the-art review. IEEE Trans. Cloud Comput. **7**(3), 677–692 (2017)
11. Yi, B.; Wang, X.; Li, K.; Huang, M.; et al.: A comprehensive survey of network function virtualization. Comput. Netw. **133**, 212–262 (2018)
12. Rodriguez, M.A.; Buyya, R.: Container-based cluster orchestration systems: a taxonomy and future directions. Softw. Pract. Exp. **49**(5), 698–719 (2019)
13. Casalicchio, E.: Container orchestration: a survey. In: Systems Modeling: Methodologies and Tools, pp. 221–235 (2019)
14. Barakabitze, A.A.; Ahmad, A.; Mijumbi, R.; Hines, A.: 5g network slicing using SDN and NFV: a survey of taxonomy, architectures and future challenges. Comput. Netw. **167**, 1–40 (2020)
15. IBM desktop virtualization system. https://www.ibm.com/cloud/learn/virtualization. Accessed 07 Nov 2019
16. Bugnion, E.; Devine, S.; Rosenblum, M.; Sugerman, J.; Wang, E.Y.: Bringing virtualization to the x86 architecture with the original vmware workstation. ACM Trans. Computer Syst. (TOCS) **30**(4), 1–51 (2012)
17. Bhardwaj, A.; Rama Krishna, C.: Efficient multistage bandwidth allocation technique for virtual machine migration in cloud computing. J. Intell. Fuzzy Syst. **35**(5), 5365–5378 (2018)
18. Mavridis, I.; Karatza, H.: Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing. Futur. Gener. Comput. Syst. **94**, 674–696 (2019)
19. Cerny, T.; Donahoo, M.J.; Trnka, M.: Contextual understanding of microservice architecture: current and future directions. ACM SIGAPP Appl. Comput. Rev. **17**(4), 29–45 (2018)
20. Lxc an operating system level container virtualization technique. https://linuxcontainers.org/lxc/introduction/. Accessed 03 Dec 2019
21. Docker a containerization technology that enables the creation and use of linux containers. https://www.docker.com/resources/what-container. Accessed 03 Dec 2019
22. Running containers with rkt virtualization technology. https://coreos.com/rkt/docs/latest/. Accessed 01 May 2019.
23. Felter, W.; Ferreira, A.; Rajamony, R.; Rubio, J.: An updated performance comparison of virtual machines and linux containers, pp. 171–172 (2015)
24. Linpack benchmark for distributed-memory computers. https://netlib.org/benchmark/hpl/. Accessed 5 June 2018
25. Stream: Sustainable memory bandwidth in high performance computers. https://www.cs.virginia.edu/stream/. Accessed 20 June 2018
26. Fio-flexible i/o tester. https://linux.die.net/man/1/fio. Accessed 16 June 2018

27. Morabito, R.; Kjällman, J.; Komu, M.: Hypervisors vs. lightweight virtualization: a performance comparison. In: Proceedings of IEEE International Conference on Cloud Engineering, pp. 386–393 (2015)
28. Y-cruncher: a multi-threaded pi-program. http://www.numberworld.org/y-cruncher/. Accessed 08 May 2019
29. Bonnie++ benchmark. https://www.coker.com.au/bonnie++/. Accessed 15 June 2018
30. Kozhirbayev, Z.; Sinnott, R.O.: A performance comparison of container-based technologies for the cloud. Future Gener. Comput. Syst **68**, 175–182 (2017)
31. Geekbench: cross-platform processor benchmark. https://www.geekbench.com/. Accessed 2 June 2018
32. Xie, X.-L.; Wang, P.; Wang, Q.: The performance analysis of docker and rkt based on kubernetes. In: Proceedings of IEEE 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), pp. 2137–2141 (2017)
33. Graph500 a data-intensive applications. https://graph500.org/. Accessed 12 March 2019
34. Containers on google cloud platform. https://cloud.google.com/containers/. Accessed 24 July 2018
35. Dua, R.; Raja, A.R.; Kakadia, D.: Virtualization vs containerization to support paas. In: Proceedings of IEEE 8th International Conference on Cloud Engineering (IC2E), pp. 610–614 (2014)
36. Yadav, A.K.; Garg, M.: Docker containers versus virtual machine-based virtualization. In: Proceedings of Springer International Conference on Emerging Technologies in Data Mining and Information Security, Singapore, pp. 141–150 (2019)
37. Luo, J.; Yin, L.; Hu, J.; Wang, C.; Liu, X.; Fan, X.; Luo, H.: Container-based fog computing architecture and energy-balancing scheduling algorithm for energy iot. Futur. Gener. Comput. Syst. **97**, 50–60 (2019)
38. Hale, J.S.; Li, L.; Richardson, C.N.; Wells, G.N.: Containers for portable, productive, and performant scientific computing. Comput. Sci. Eng. **19**(6), 40–50 (2017)
39. Overlay. https://coreos.com/blog/announcing-rkt-0.5/. Accessed 08 Nov 2019
40. Docker swarm container orchestration system. https://www.docker.com/blog/scale-testing-docker-swarm-30000-containers/. Accessed 20 Jan 2020
41. Kovács, J.; Kacsuk, P.; Emődi, M.: Deploying docker swarm cluster on hybrid clouds using occopus. Adv. Eng. Softw. **125**, 136–145 (2018)
42. Kubernetes container orchestration system. https://kubernetes.io/docs/setup/best-practices/cluster-large/. Accessed 20 Jan 2020
43. Truyen, E.; Van Landuyt, D.; Preuveneers, D.; Lagaisse, B.; Joosen, W.: A comprehensive feature comparison study of open-source container orchestration frameworks. Appl. Sci. **9**(5), 931–938 (2019)
44. Guerrero, C.; Lera, I.; Juiz, C.: Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications. J. Supercomput. **74**(7), 2956–2983 (2018)
45. Liu, S.; Liu, Z.: Introduction to linux and command line tools for bioinformatics. In: Bioinformatics in Aquaculture: Principles and Methods, pp. 1–29 (2017)
46. Docker stats display a live stream of container resource usage statistics. https://docs.docker.com/engine/reference/commandline/stats/. Accessed 24 Nov 2019
47. Monitoring docker container metrics using cadvisor. https://prometheus.io/docs/guides/cadvisor/. Accessed 20 Nov 2019
48. Prometheus an open-source systems monitoring and alerting toolkit. https://prometheus.io/docs/introduction/overview/. Accessed 24 Nov 2019
49. Sensu container monitoring framework. http://sensu-plugins.io/. Accessed 20 Nov 2019
50. Sysdig container kubernetes monitoring, alerting, and troubleshooting tool. https://sysdig.com/platform/. Accessed 15 Nov 2019
51. Karn, R.R.; Kudva, P.; Elfadel, I.A.M.: Dynamic autoselection and autotuning of machine learning models for cloud network analytics. IEEE Trans. Parallel Distrib. Syst. **30**(5), 1052–1064 (2018)
52. Rovnyagin, M.M.; Hrapov, A.S.; Guminskaia, A.V.; Orlov, A.P.: Ml-based heterogeneous container orchestration architecture. In: Proceedings of IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), pp. 477–481 (2020)
53. Sultan, S.; Ahmad, I.; Dimitriou, T.: Container security: Issues, challenges, and the road ahead. IEEE Access **7**, 52976–52996 (2019)
54. Gummaraju, J.; Desikan, T.; Turner, Y.: Over 30% of official images in docker hub contain high priority security vulnerabilities. Technical Report, Banyan Ops (2015)
55. Shu, R.; Gu, X.; Enck, W.: A study of security vulnerabilities on docker hub. In: Proceedings of the Seventh International Conference on Data and Application Security and Privacy, pp. 269–280. ACM (2017)
56. Xie, X.; Huang, T.; Guo, Z.: Research on the security protection scheme for container-based cloud platform node based on blockchain technology. In: Proceedings of International Conference on Pioneering Computer Scientists, Engineers and Educators, pp. 24–32 (2018)
57. Top open source tools for docker security. https://techbeacon.com/security (2020). Accessed 1 Sept 2020
58. Morabito, R.: Virtualization on internet of things edge devices with container technologies: a performance evaluation. IEEE Access **5**, 8835–8850 (2017)
59. Kaur, K.; Dhand, T.; Kumar, N.; Zeadally, S.: Container-as-a-service at the edge: trade-off between energy efficiency and service availability at fog nano data centers. IEEE Wirel. Commun. **24**(3), 48–56 (2017)
60. Alves, M.P.; Delicato, F.C.; Santos, I.L.; Pires, P.F.: Lw-coedge: a lightweight virtualization model and collaboration process for edge computing. World Wide Web **23**(2), 1127–1175 (2020)
61. Haji, L.M.; Zeebaree, S.; Ahmed, O.M.; Sallow, A.B.; Jacksi, K.; Zeabri, R.R.: Dynamic resource allocation for distributed systems and cloud computing. TEST Eng. Manag. **83**, 22417–22426 (2020)
62. Reclaiming or rebalancing the unused and overallocated vSphere virtual machines. https://docs.bmc.com/docs/btco113/reclaiming-the-unused-and-overallocated-azure-virtual-machines-785283461.html (2020). Accessed 02 Sept 2020
63. Hu, Y.; Zhou, H.; de Laat, C.; Zhao, Z.: Ecsched: Efficient container scheduling on heterogeneous clusters. In: European Conference on Parallel Processing, pp. 365–377 (2018)
64. Hu, Y.; Zhou, H.; Laat, C.; Zhao, Z.: Concurrent container scheduling on heterogeneous clusters with multi-resource constraints. Futur. Gener. Comput. Syst. **102**, 562–573 (2020)
65. Ludwig, U.L.; Xavier, M.G.; Kirchoff, D.F.; Cezar, I.B.; De Rose, C.A.: Optimizing multi-tier application performance with interference and affinity-aware placement algorithms. Concurr. Comput. Pract. Exp. **31**(18), 1–16 (2019)
66. Chen, W.-Y.; Ye, K.-J.; Lu, C.-Z.; Zhou, D.-D.; Xu, C.-Z.: Interference analysis of co-located container workloads: a perspective from hardware performance counters. J. Comput. Sci. Technol. **35**, 412–417 (2020)
67. Kim, S.; Kim, Y.: Toward interference-aware gpu container co-scheduling learning from application profiles. In: Proceedings of IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), pp. 19–23 (2020)
68. Zhao, Y.; Xia, N.; Tian, C.; Li, B.; Tang, Y.; Wang, Y.; Zhang, G.; Li, R.; Liu, A.X.: Performance of container networking tech-

nologies. In: Proceedings of Workshop on Hot Topics in Container Networking and Networked Systems, pp. 1–6 (2017)
69. Suo, K.; Zhao, Y.; Chen, W.; Rao, J.: An analysis and empirical study of container networks. In: Proceedings of IEEE International Conference on Computer Communications, pp. 189–197 (2018)
70. Abbasi, U.; Bourhim, E.H.; Dieye, M.; Elbiaze, H.: A performance comparison of container networking alternatives. IEEE Netw. **33**(4), 178–185 (2019)
71. Mijumbi, R.; Serrat, J.; Gorricho, J.-L.; Bouten, N.; De Turck, F.; Boutaba, R.: Network function virtualization: state-of-the-art and research challenges. IEEE Commun. Surv. Tutor. **18**(1), 236–262 (2015)
72. Hawilo, H.; Jammal, M.; Shami, A.: Exploring microservices as the architecture of choice for network function virtualization platforms. IEEE Netw. **33**(2), 202–210 (2019)