



Comparative Analysis of Virtualization Methods in Big Data Processing

Gleb I. Radchenko¹ , Ameer B. A. Alaasam¹, Andrei N. Tchernykh^{1,2} 

© The Authors 2019. This paper is published with open access at SuperFri.org

Cloud computing systems have become widely used for Big Data processing, providing access to a wide variety of computing resources and a greater distribution between multi-clouds. This trend has been strengthened by the rapid development of the Internet of Things (IoT) concept. Virtualization via virtual machines and containers is a traditional way of organization of cloud computing infrastructure. Containerization technology provides a lightweight virtual runtime environment. In addition to the advantages of traditional virtual machines in terms of size and flexibility, containers are particularly important for integration tasks for PaaS solutions, such as application packaging and service orchestration. In this paper, we overview the current state-of-the-art of virtualization and containerization approaches and technologies in the context of Big Data tasks solution. We present the results of studies which compare the efficiency of containerization and virtualization technologies to solve Big Data problems. We also analyze containerized and virtualized services collaboration solutions to support automation of the deployment and execution of Big Data applications in the cloud infrastructure.

Keywords: Big Data, visualization, containerization, cloud computing, Xen, KVM, Docker, orchestration.

Introduction

Cloud computing systems have become widely used for implementation of Big Data processing tasks. Clouds rely on virtualization technology to achieve the elasticity of shared computing resources.

Virtual Machines (VMs) underlie the cloud computing infrastructure layer, providing virtual operating systems. Virtualization is a combination of hardware and software solutions that support the creation and operation of virtual versions of devices or resources, such as servers, storage devices, networks, or operating systems. The virtualization platform allows one to divide the physically unified hardware system into a logical set of independent computing resources [55]. Virtualization of computing resources allowed to solve the problem of increasing the efficiency of scheduling in cluster computing systems, by presenting their resources in the format of independent virtual machines [6]. Virtual machines provide isolation of the file system, memory, network connections, and system information [92]. But the use of virtual machines is associated with large overheads [38, 101], which can significantly limit the performance of I/O systems and efficiency of the computational resources.

The containerization technology has significantly advanced recently. It is based on the concept of limiting the amount of resources that are provided to an application by the computational node. The container provides a runtime environment for the application at the operating system level [13], reducing the overhead compared to a virtual machine.

Virtual machines and containers are virtualization technologies, but they differ in goals and functions. Containers can be viewed as a flexible tool for packaging, delivering and orchestrating both applications and software infrastructure services. They allow you to focus on a portable way to increase compatibility [73], while still using the principles of operating system virtualization.

¹South Ural State University, Chelyabinsk, Russian Federation

²CICESE Research Center Ensenada, Mexico

On the other hand, virtual machines are associated with the distribution and management of computational infrastructure.

In this paper we would provide an overview of the current state of virtualization and containerization approaches and technologies in the context of Big Data tasks solution. The rest of the paper is organized as follows. In Section 1 we would analyze the basics of virtualization technologies, together with the brief overview of most popular open-source virtualization solutions: Xen and KVM hypervisors. Section 2 is devoted to the overview of containerization technologies. We would analyze the key features of containerization approach and take a look at the architecture of the most popular containerization solutions and frameworks, such as Docker. In the Section 3 we would overview the main results on the comparison of containerization and virtualization solutions. Section 4 would be devoted to the overview of container orchestration technologies. In the last Section, we would provide conclusions on the performed analysis.

1. Virtualization Technologies

Virtualization was developed for abstracting the hardware and system resources to provide simultaneous execution of several operating systems on a single hardware platform [6].

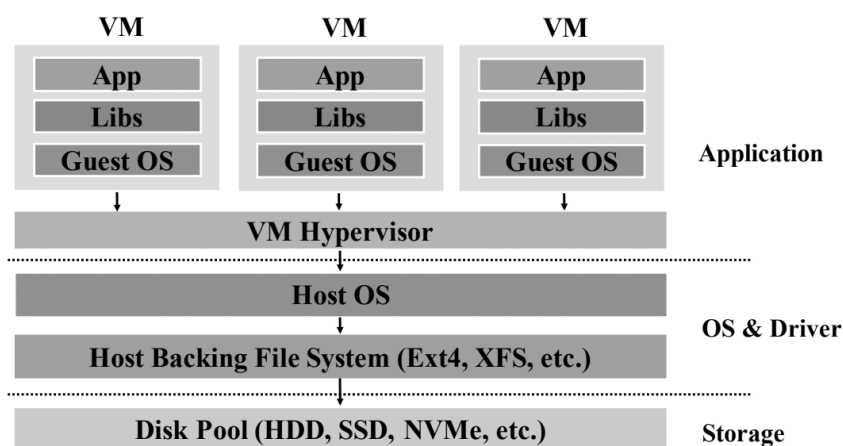


Figure 1. The architecture of the virtual machine hypervisor (based on [9])

Virtual Machine Hypervisor technology, also called Virtual Machine Monitor, has a long history since the 1960s and was widely used before the era of cloud computing. As shown in Fig. 1, a virtual machine hypervisor (for example, Xen, KVM, VMware, etc.) is software that provides a virtual platform, so several guest operating systems can run on one system server. The hypervisor runs as a middleware between the virtual machine and the OS. Each virtual machine has its own guest OS.

There are several approaches to implementing a virtual machine hypervisor. So, *full virtualization* [98] is aimed at hardware emulation. In this case, a non-modified OS is used inside a virtual machine, and the hypervisor controls the execution of privileged operations of the guest OS. *Paravirtualization* requires modifications of the virtualized OS and coordination of operations between the virtual OS and the hypervisor [98]. Usage of paravirtualization improves the performance with respect to full virtualization by performing most of the operations in the host OS.

Virtual instances use isolated, large files on their host as guest images that store the file system and run the virtual machine as one large process on the host. This approach leads to some performance degradation. A complete image of a guest OS, together with the binary files and libraries needed for applications, are required for each virtual machine. This leads to significant overhead on the required disk space, and also leads to additional RAM requirements when executing virtual machines on the server. It also causes performance issues, such as slow image startup. In addition, multiple owner clouds require sharing of disk space and processor cycles. In a virtualized environment, this should be managed in such a way that the underlying platform and infrastructure can be shared in a safe but compatible way [72].

1.1. Xen Hypervisor

Created by researchers at Cambridge University [6], Xen is a VM hypervisor that is operating on a physical machine in the most privileged processor mode compared to any other software. The Xen hypervisor is responsible for memory management, processor resources scheduling and running a privileged domain of a physical machine (referred to as Dom0 in Xen terminology), which has direct access to hardware devices.

Dom0 domain starts at the launch of a physical machine and is usually implemented as a modified Linux, NetBSD, or Solaris OS [102]. One can manage the hypervisor and run other non-privileged guest domains (DomU) from this domain. Xen implements paravirtualization approach, so the guest OS in the DomU domain should be modified to access hardware through the Dom0 using paravirtualized drivers and an interdomain channel [69]. Figure 2 shows the operation of the Xen hypervisor and its domains.

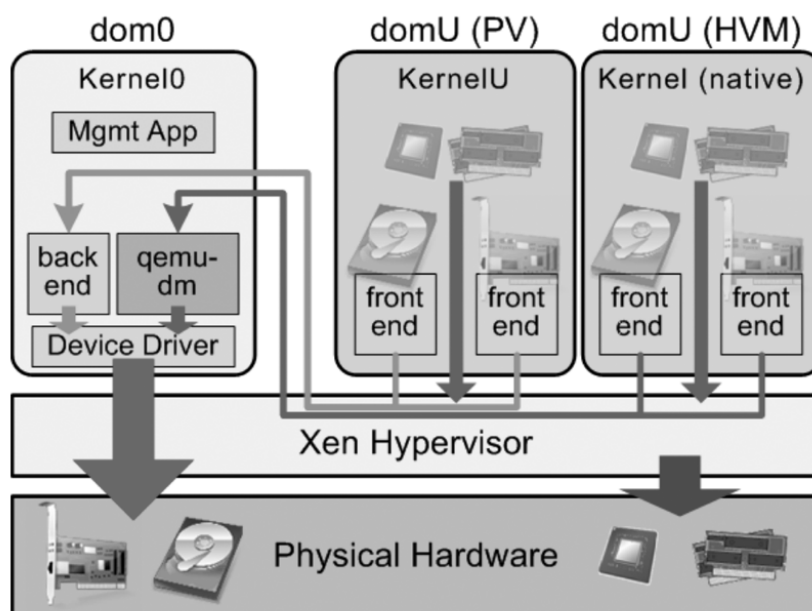


Figure 2. Xen hypervisor architecture (based on [69])

1.2. KVM Hypervisor

Kernel-based Virtual Machine (KVM) is a virtualization infrastructure integrated into the Linux kernel. This hypervisor was first developed by Qumranet company in 2006 [42]. Instead of creating a hypervisor from scratch, the Linux kernel was used as the KVM basis. It relies on hardware-assisted virtualization support by the host processors. KVM includes a loadable kernel module (`kvm.ko`), providing the basic virtualization infrastructure and a processor module (either `kvm-intel.ko`, or `kvm-amd.ko`).

In KVM virtualization model, virtual machines are created by opening a `/dev/kvm` device node. KVM uses a slightly modified QEMU emulator (called `qemu-kvm`) to create virtual machine instances. Each guest virtual machine is implemented as a standard Linux process, managed by a standard Linux scheduler. In addition to the two standard execution modes (kernel mode and user mode), KVM adds a third mode: guest mode, which has its own kernel and user modes that the hypervisor does not control. The modified QEMU emulator also handles I/O hardware emulation, as shown in Fig. 3.

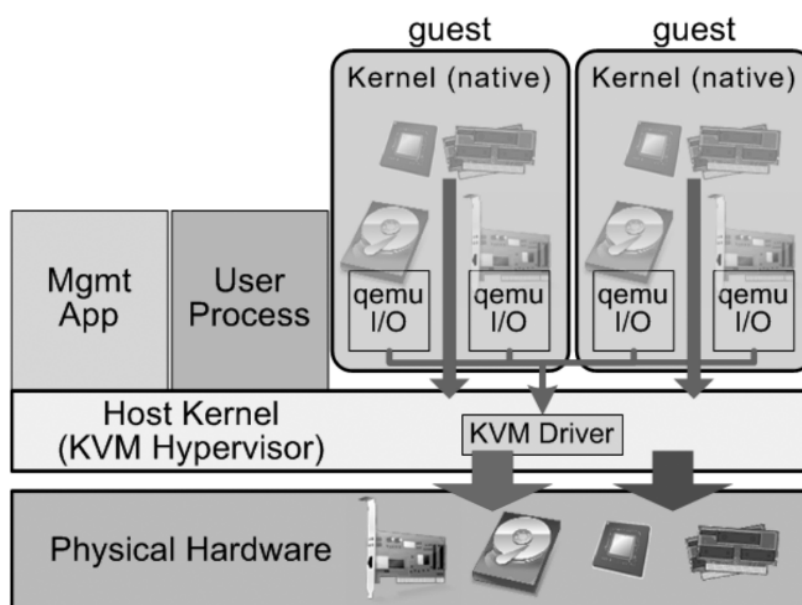


Figure 3. KVM hypervisor architecture (based on [69])

2. Containerization Technologies

Unlike full virtualization and paravirtualization, the OS-level virtualization approach does not require a hypervisor. It implies that the OS is changed to ensure that several OS copies are able to be executed on the same machine [98]. Linux OS-based virtualization is called container-based virtualization [101].

A container is a packaged, standalone, deployable set of application components, which may also include middleware and business logic as binary files and libraries for running applications [85] (see Fig. 4). Containers are the building blocks of OS-level virtualization that allows isolated virtual environments without the need of hypervisors. These virtual structures are in-

dependent of each other, but share the same underlying operating system (i.e., the kernel and device drivers) [17].

Docker today is one of the most well-known platforms for organizing solutions based on container technologies [2]. Such platforms turn containers into a convenient way to package and deliver applications [13].

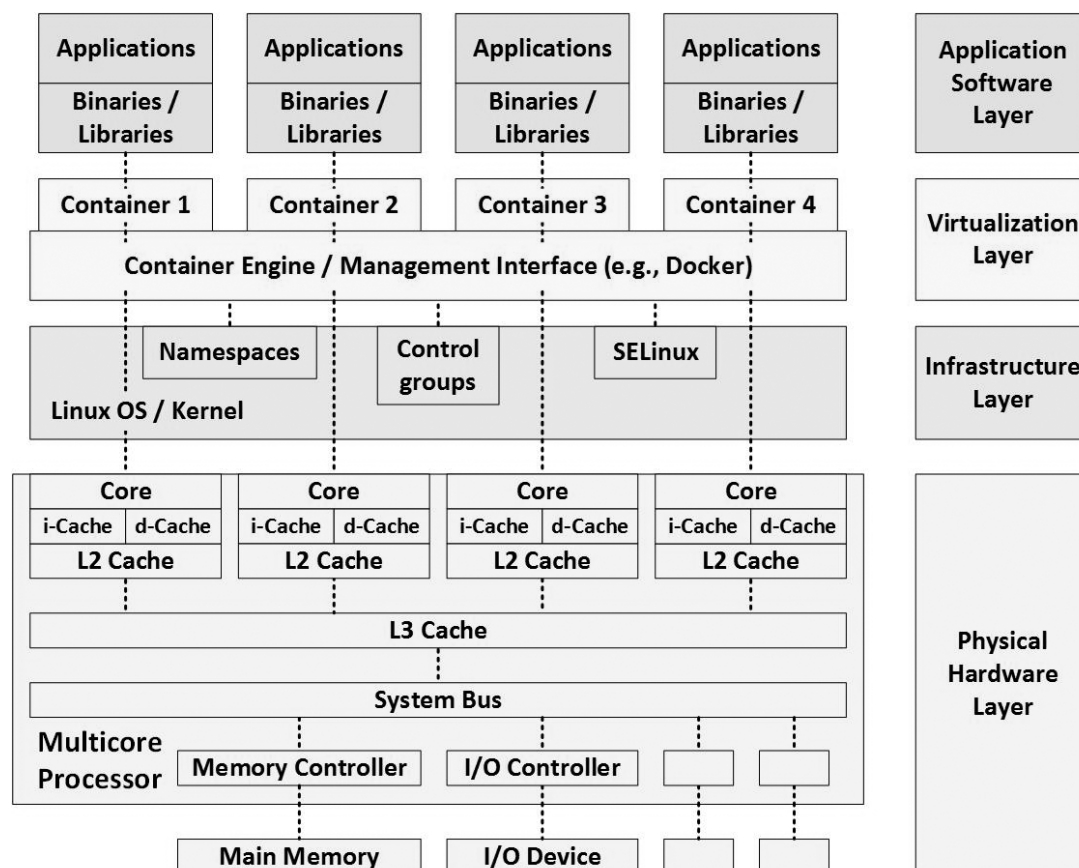


Figure 4. Container-based virtualization architecture (based on [31])

2.1. Namespaces and Cgroups

In modern Linux distributions, the LXC virtualization project (Linux containers) implements kernel mechanisms such as *namespaces* and *control groups* (*cgroups*) to isolate processes on a shared operating system [85].

Namespace isolation allows you to separate process groups. This ensures that they cannot see resources in other groups. Different namespaces are used to isolate processes, network interfaces, access interposes communication, mount points, or to isolate kernel identifiers and version identifiers.

On the other hand, cgroups control and restrict access to resources for groups of processes by enforcing CPU resource limits, accounting and isolation, for example, limiting the memory available to a specific container. This provides isolation between applications on the host. It also limits containers in multi-tenant host environments. Control groups allow containers to share available hardware resources and establish restrictions as needed.

At the same time, the authors of [104] explore the performance of the cgroups technology using the example of LinkedIn, where this technology is used as the basis for management of the distributed computing resources. They indicate the following key technology limitations:

- 1) memory is not reserved for cgroups (as opposed to virtual machines);
- 2) both the shared memory and the cache pages are in the common memory pool, while the former can displace the latter;
- 3) the OS can take over a page cache from any of the cgroups.

2.2. Key Containerization Technologies

The simplicity of tools and ease of creation and management of containerized environment made *Docker* a popular open source project. Docker containers can run only Linux processes, but one can use Linux, Windows or MacOS machines as a host. Docker containers provided greater efficiency in software development, but orchestration tools such as Docker Swarm [55] or Kubernetes [90] are required for enterprise use.

Java containers such as Jetty [88], Tomcat [4], Wildfly [76], and Spring Boot [71] are examples of container technologies that provide usage of standalone Java applications. The result of such systems are containerized Java applications that can run without the need for an external Java environment.

LXD is a container platform from Canonical [16]. LXD containers are created and operated using the same tools as traditional virtual machines, but they can provide high performance at runtime that matches the performance of container solutions. Unlike the Docker, LXD container management does not require additional orchestration systems, such as Swarm or Kubernetes. LXD is much closer to the full operating environment of the virtual machine hypervisor, including the virtualization of network interfaces and data storage interfaces. LXD containers in this case are much closer to full-featured virtual machines. For example, it is possible to run multiple Docker containers within LXD [84].

OpenVZ (Open Virtuozzo) [64] is one of the oldest Linux container platforms still in use today, with roots dating back to 2005. Before OpenVZ the Linux kernel had no means to create any sort of containerization, apart from the `chroot()` functionality that allowed a process to be run using a different view of the filesystem. LXC itself is a spiritual successor of OpenVZ. While OpenVZ is still around, today LXC is the tool of choice for many who wish to run a full operating system in a container [70].

Rkt [75] is a container technology introduced in the CoreOS platform to address security vulnerabilities in earlier versions of Docker. In 2014, CoreOS published the App Container (appc) specification to stimulate innovation in container space, which spawned a number of open source projects. It is necessary to clarify that Docker's early vulnerabilities have already been resolved, and Docker v1.11 implements the Open Container Initiative (OCI) standard [89] supported by RedHat, Google, AWS, VMware and CoreOS, thus ensuring compatibility with rkt.

Another consequence of the implementation of the Open Container Initiative standard was the *CRI-O* project [21], launched in 2016 with the participation of such companies as Red Hat, Intel, SUSE, Hyper and IBM. CRI-O allows users to run and manage any containers that are compatible with OCI directly from the Kubernetes platform without additional code or tools. From the end user's point of view, both the Docker and CRI-O implement the Kubernetes Container Runtime Interface (CRI) and implement the same functionality (loading and launching containers). CRI was built mainly to ensure that the Kubernetes platform was not heavily

dependent on Docker. Before this standard was adopted, Kubernetes was developed based on assumptions specific to the docker architecture, including such variables as paths to volumes, the internal architecture of the container, containers images storage specification, etc.

Windows Containers [54] technology was introduced along with the launch of Windows Server 2016. Microsoft has made significant improvements to the architecture of the Windows operating system to ensure the operation of container technologies, working closely with Docker to ensure the seamless operation of Docker container management tools in the Microsoft infrastructure. Currently, a number of works are underway to optimize the size of container images. Their work is provided in Windows 10, Server 2019 and Microsoft Azure cloud platform.

While discussing virtualization and containerization technologies we should mention the Unikernels approach. *Unikernels* are single-purpose appliances that are compile-time specialised into standalone kernels, sealed against modification when deployed to a cloud platform and act as separate software components [48]. The final application consists of a set of executable unikernels working together as a distributed system [49]. Unikernels provide optimization of the resources required by the container. One can identify the dependencies of the runtime of the application and package them into a single image, providing only the functionality of the OS that is necessary at the application runtime. Unlike Docker containers, unikernels can load and run completely independently, without a host operating system or external libraries, while Docker relies on external resources and the host environment to run. Unikernels can reduce complexity, improve portability, and reduce the attack surface of applications, but they require new development and deployment tools that are still not well developed.

2.3. Docker

Docker is a container solution based on LXC approach [22]. Docker images are made up of file systems arranged in layers, one above the other, like a Linux virtualization stack using LXC mechanisms. A container daemon, called *dockerd*, starts containers as application processes. It plays a key role as the root of the user process tree.

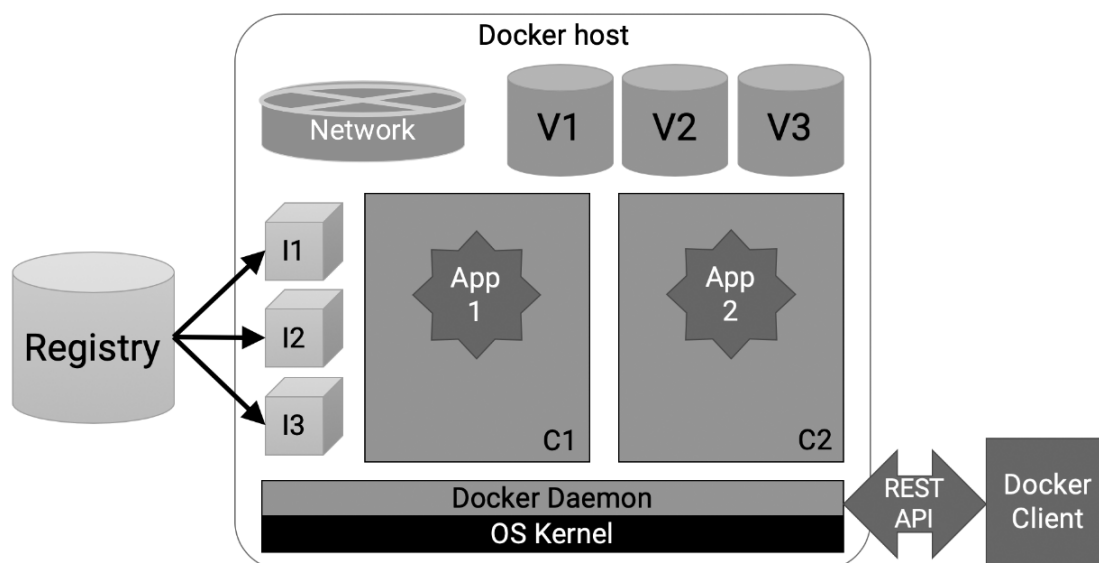


Figure 5. Docker deployment architecture on a node

Docker provides a complete infrastructure for containerization, including:

1. *Docker Engine* (kernel) consisting of a daemon, a REST API and a client.
2. *Orchestration mechanisms*: Docker Machine, Docker Swarm, Docker Compose.
3. *Installation packages* for desktop and cloud platforms: Linux, MacOS, Windows, Microsoft Azure, Amazon Web Services.
4. *Online services*: Docker HUB, CS Engine, Trusted Registry, Docker Cloud, Docker Store.

The Docker system deployed on the node is shown in Fig. 5, where:

- *Docker Daemon* is a management system, that manages containers, images, volumes, virtual networks.
- *C1..C2* are executable Docker containers, representing one or several processes running in an isolated environment. This environment provides no access to external processes, has its own root file system, network configuration (including hostname, IP-address, etc.). A container is executed within the framework of limitations on computational, memory, network and other I/O resources.
- *I1..I3* are images of Docker containers: read-only file system presets, containing the OS files, applications, all the necessary libraries and dependencies, except for the OS kernel itself. A container image is a way to distribute applications. Container images have a layered structure consisting of several images built on top of the base image. A typical division of an image of a container into layers may include from top to bottom: a modifiable container image for applications, basic images (for example, Apache and Emacs), a Linux image (for example, Ubuntu), and a *rootfs* kernel *image* (see Fig. 6).
- *V1..V3* are virtual volumes that provide long-term storage of data outside the container.
- *Docker Network* — a virtual network infrastructure that provides communication between containers, containers and the host, containers and the outside world.
- *REST API* is an API for Docker Daemon management.
- *Docker Client* is a Command Line Interface that provides management for the Docker infrastructure.

The Docker platform is gradually gaining popularity in the field of scientific problems associated with the Big Data processing. This is due to the fact that the Docker platform provides a single mechanism for containerized applications execution on the basis of a distributed computational environment, while simultaneously providing the necessary interfaces for network ports, volumes, etc., allowing different system users to work within the standardized computing environment [30].

2.4. NVIDIA Container Runtime

Big Data processing tasks often involve GPU resources in their solution for implementing parallel computing. In this regard, a number of attempts have been made to introduce virtualized graphics processors into virtual machines, including such approaches as GViM [35], gVirtuS [34] and vCUDA [81]. These approaches are based on creating copies of the CUDA API for virtualizing graphics processors and providing them to virtual machines. As a part of the rCUDA solution [26], the technology of remote graphics processors usage was proposed. However, these methods have their drawbacks, as they degrade the performance of the graphics processor during the virtualization process [41]. Moreover, these methods provide only a part of the CUDA API [40].

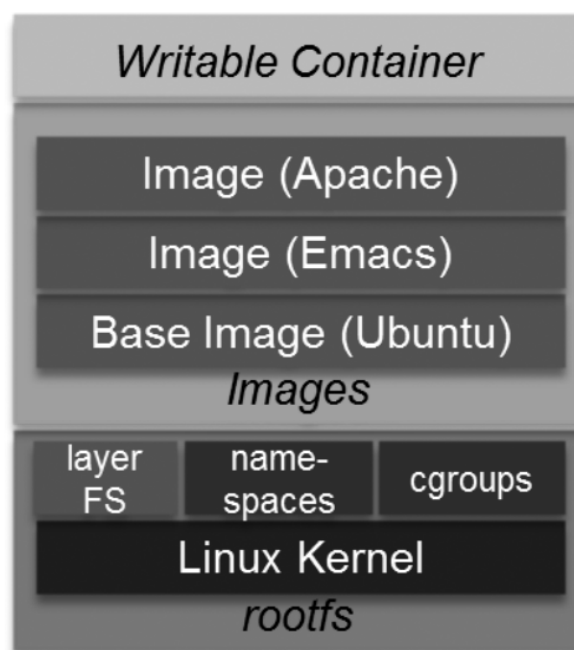


Figure 6. Container image structure in the Docker system (based on [68])

In 2016 NVIDIA Corporation proposed a solution called NVIDIA Docker [57], which differs from the approaches described above. NVIDIA Docker is a utility that makes it easy to use an NVIDIA GPU inside a Docker container. NVIDIA Docker contains two executable programs: `nvidia-docker` and `nvidia-docker-plugin`.

`nvidia-docker` is a shell on top of the Docker [59], which intercepts user commands to use the `nvidia-docker` command instead of the original `docker` command. The role of this command is to interpret and modify user commands with their subsequent transfer to the Docker command interface. `nvidia-docker` captures only the `run` and `create` commands, the rest of the commands are translated directly to the Docker. `nvidia-docker` checks whether the launched image uses the CUDA API using `com.nvidia.volumes.needed` and `com.nvidia.cuda.version` Docker labels, which specify the required CUDA versions. `nvidia-docker` uses data from these labels to determine the number and location of installed graphics devices and links them using the `--device` option. In addition, it links the correct versions of the GPU drivers using the `--volume` option, which is directly related to the `nvidia-docker-plugin`.

The `nvidia-docker-plugin` is an add-on designed to facilitate the deployment of containers that support GPUs. It acts as a daemon, identifies driver files, GPU devices, and responds to volume mount requests from the Docker daemon [60]. The purpose of the `nvidia-docker-plugin` is to check the existence of the NVIDIA GPU and CUDA API, as well as to provide the necessary versions of the binaries and libraries to the running container. The version of the CUDA API requested by the `nvidia-docker` command is provided via the `nvidia-docker-plugin` in the volume with the corresponding name. When the container completes its work, the driver volume shuts down.

The evaluation of the NVIDIA Docker approach shows that the performance of containerized GPU-accelerated applications is no different from the performance of the same applications

using GPUs outside the container [33]. NVIDIA Docker is successfully used to solve problems in machine learning, implemented on top of containerized infrastructures [46].

This approach is developed further into the NVIDIA Container Runtime [58] solution, which removes some of the limitations of the `nvidia-docker` project, including:

- support for the most common container technologies, such as LXC and CRI-O [21];
- compatibility with docker ecosystem tools, such as compose, for management of applications that use GPUs and consist of several containers;
- support of GPUs as resources in Kubernetes and Swarm container orchestration systems.

3. Comparison of Containerization and Virtualization Solutions

The virtual machine hypervisor emulates the hardware on top of which guest operating systems are running. Containers provide virtualization solutions at the level of the operating system: each guest OS uses the same kernel (and in some cases other parts of the OS) as the host. Such difference gives an advantage to containers approach: they are smaller and more compact than hypervisor guest environments since they have much more in common with the host.

In this section, we study the opportunities offered by containerization and virtualization solutions and present the results which compare the efficiency of containerization and virtualization technologies to solve the practical problems.

3.1. Virtual Machines and Containers Comparison

Authors of [9, 25, 68] provide various methodologies for comparing containerization and virtualization technologies. General characteristics and approaches of comparing these technologies are presented in Tab. 1.

The following advantages have led to the widespread use of virtualization via containers [31]:

1. **Hardware costs.** Virtualization via containers decreases hardware costs by enabling consolidation. It enables concurrent software to take advantage of the true concurrency provided by a multicore hardware architecture.
2. **Reliability and robustness.** The modularity and isolation provided by VMs improve reliability, robustness, and operational availability by localizing the impact of defects to a single VM and enabling software failover and recovery.
3. **Scalability.** A single container engine can efficiently manage large numbers of containers, enabling additional containers to be created as needed.
4. **Spatial isolation.** Containers support lightweight spatial isolation by providing each container with its own resources (e.g., core processing unit, memory, and network access) and container-specific namespaces.
5. **Storage.** Compared with virtual machines, containers are lightweight with regard to storage size. The applications within containers share both binaries and libraries.
6. **Performance.** Compared with virtual machines, containers increase performance (throughput) because they do not emulate the underlying hardware. Note that this advantage is lost if containers are hosted on virtual machines (i.e., when using a hybrid virtualization architecture).
7. **Real-time applications.** Containers provide more consistent timing than virtual machines, although this advantage is lost when using hybrid virtualization.

Table 1. Comparison of virtual machines and containers (based on [9, 25, 68])

Characteristic	Virtual machines	Containers
Products	VMware, Xen, KVM, ...	Docker, rkt, LCX, OpenVZ, ...
Guest OS	Each virtual machine is executed on the basis of its own OS loaded into its own block of RAM within the framework of virtual hardware.	All containers are executed on the basis of the OS kernel of the host machine.
Process management	Virtual machine hypervisor.	Namespaces, cgroups.
Isolation	Direct sharing of files or system libraries among guest and host OS is impossible.	Catalogues can be transparently shared across multiple containers.
Image size	Large image, because it includes the entire image of the base OS and all related libraries.	Smaller image size, since a common OS kernel image is used.
Start-up time	Starting a virtual machine takes a few minutes.	Start-up time can be a few seconds.
Process of loading and execution	After the standard boot process on the host, each virtual machine is represented as a separate process.	Applications can be started in containers directly or via an initial daemon known to the container, for example <i>dockerd</i> . They appear as normal processes on the host.

8. **Continuous integration.** Containers support continuous development processes by enabling the integration of increments of container-hosted functionality.

9. **Portability.** Containers support portability from development to production environments, especially for cloud-based applications.

Although there are some disadvantages of containerization that should be addressed, especially in such cases as shared resources usage, weaker isolation, and security issues, comparing to virtual machines. Researchers and developers must address challenges and associated risks in the following areas when using containers:

1. **Shared resources.** Applications within containers share many resources including container engine, OS kernel, the host operating system, processor-internal resources (L3 cache, system bus, memory controller, I/O controllers, and interconnects) and processor-external resources (main memory, I/O devices, and networks). Such shared resources imply that software running in one container can impact software running in another container [31].
2. **Security.** Containers are not by default secure and require significant work to make them secure. One must ensure that no data is stored inside the container, container processes

are forced to write to container-specific file systems, the container's network namespace is connected to a specific, private intranet, and the privileges of container services are minimized (e.g., non-root if possible). There were several well-known flaws, those allowed attackers to create an application that would be able to escape the container and attack the host system. One of such vulnerabilities (recently patched) allowed a malicious container to (with minimal user interaction) overwrite the host `runc` binary and thus gain root-level code execution on the host [79].

In view of the above many cloud administrators and software architects suggest running a container on top of a VM to enhance security. Moreover this technic is implemented for easier management and upgrade of the system, as well as to overcome hardware and software incompatibility with the physical server. But all the aforementioned benefits come with a performance cost [51].

3.2. Comparative Effectiveness of Virtualization Technologies

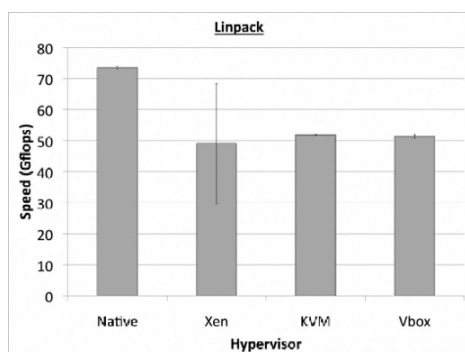
Authors of [103] compare hypervisor models with full virtualization and paravirtualization, such as Xen [6], KVM [44], VirtualBox [66] and VMWare ESX [95]. A comparison of characteristics of analyzed virtualization platforms is provided in Tab. 2. According to the test results, which included a comparison of the performance of the Linpack test, performing a fast Fourier transform, evaluating bandwidth and latency of network connections, as well as running an OpenMP program, the KVM platform (followed by VirtualBox) was recognized as the leader in performance (see Fig. 7). Unfortunately, the results of the effectiveness of VMWare ESX are not published in the article due to the limitations of the license agreement.

Table 2. Comparison of characteristics of virtualization platforms (based on [65, 87, 91, 94])

	Xen 4.11	KVM	VirtualBox 4.1	VMWare ESXi 6.7
Paravirtualization	Yes	No	No	Yes
Supported CPU	x86, x86-64, IA64	x86, x86-64, IA64, PPC	x86, x86-64	x86, x86-64
Host OS	Linux, UNIX	Linux, UNIX	Windows, Linux, UNIX	Proprietary UNIX
Guest OS	Windows, Linux, UNIX	Windows, Linux, UNIX	Windows, Linux, UNIX	Windows, Linux, UNIX
CPUs per host (x86)	4095	4095	No Limit	768
Memory per host	16 TB	16 TB	1 TB	16 TB
License	GPL	GPL	GPL/proprietary	Proprietary

The mechanisms of Xen's complete and paravirtualization models were investigated in [28]. The research results show that when using the full virtualization approach, the overhead is at least 35 % more compared to the paravirtualization model.

Many researchers are focusing on the efficient use of virtualized resources for solving Big Data processing problems. So, the authors of [47] compare an unnamed, but widely used, commercial hypervisor with open solutions: Xen and KVM. The main analysis is a series of typical data processing tasks on the Hadoop platform [83]. The greatest differences in performance were observed in tasks related to high I/O usage, while tests related to high CPU performance demand showed smaller differences between hypervisors. It was discovered that a commercial hypervisor



(a) Linpack test results

	Xen	KVM	VirtualBox
Linpack	3	1	2
FFT	3	1	2
Bandwidth	2	3	1
Latency	3	2	1
OpenMP	2	1	3
Total Rating	13	8	9

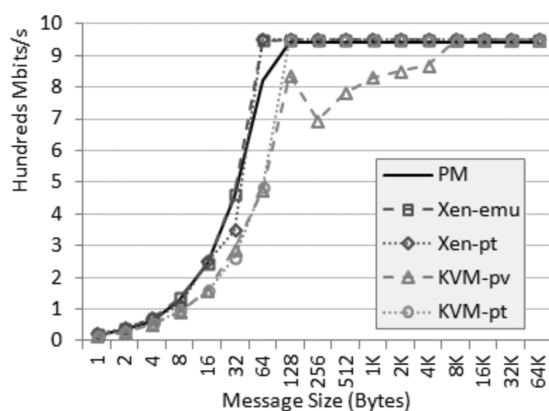
(b) The resulting rating (the smaller the better)

Figure 7. Testing of virtualization platforms (based on [103])

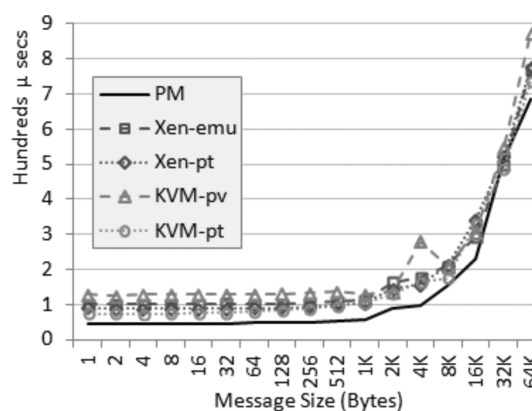
works better with disk write tasks, KVM is better for reading from disk, and Xen was better when the task required a combination of reading and writing a disk with intensive CPU calculations.

Virtualization with hardware support was tested on the basis of Xen and KVM hypervisors in the article [69] (see Fig. 8 and 9). In particular, to provide direct access to hardware devices by virtual machines, PCI transit technologies were used. The results showed that the use of hardware support technologies for virtualization can achieve low overhead costs both for performing I/O operations and for tasks that require large CPU resources.

The carried out tests showed that the network delay increases on average by 30–60 microseconds when using a Gigabit Ethernet type network and by 20–30 microseconds in InfiniBand systems. Performing tests on pure InfiniBand hardware in TCP mode gave a network delay of 20 to 130 microseconds depending on the packet size (about 25 microseconds with packets up to 2 kilobytes on average). Using RDMA reduced the delay to about 10–50 microseconds, depending on the packet size (about 10 microseconds with a packet size of up to 2 kilobytes on average). The use of virtualization, in this case, results in an increase in the response delay of 1.5–2.5 times: in TCP mode from 50 to 170 microseconds (about 65 microseconds with a packet size up to 2 kilobytes on average), in RDMA mode both for Xen and KVM the delay is from 25 to 80 microseconds, depending on the packet size (about 30 microseconds with packet sizes up to 2 kilobytes on average). Performance evaluation of an MPI application showed performance degradation of virtualized systems by 20–50 % with KVM virtualization versus Xen.



(a) Bandwidth



(b) Network latency

Figure 8. Comparison of Gigabit Ethernet type network performance on bare metal (PM), and different types of Xen and KVM virtualization (based on [69])

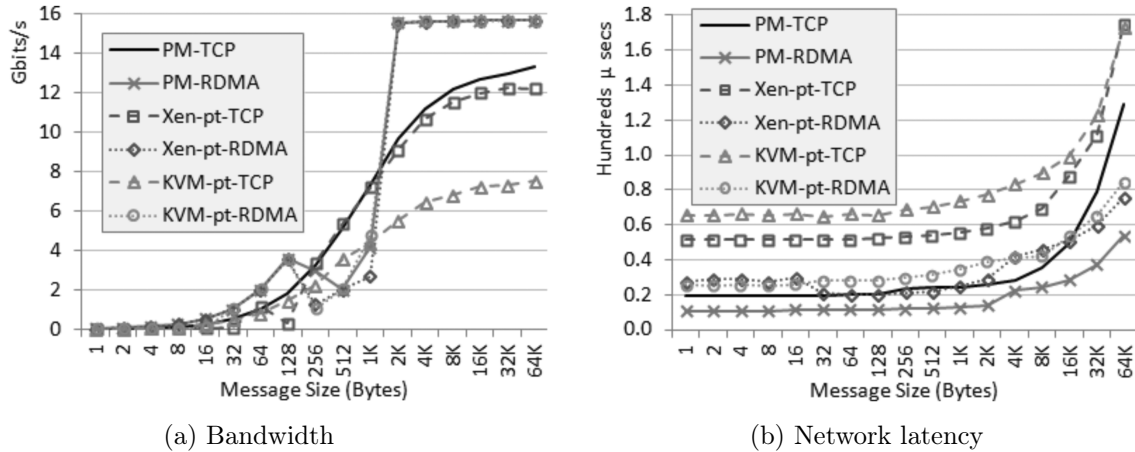


Figure 9. Comparison of an InfiniBand network performance on bare metal (PM), and different types of Xen and KVM virtualization (based on [69])

3.3. Comparison of Container and Virtual Machine Performance

The authors of [98] discuss the comparison of VMware, Xen and OpenVZ virtualization and containerization technologies from the point of view of different workloads. OpenVZ as the solution for virtualization at the OS level showed the least amount of overhead and the highest performance. Che et al in [18] also evaluated the performance of OpenVZ, Xen and KVM. Test results showed that OpenVZ has better performance, and KVM (full virtualization) has lower performance than Xen (paravirtualization).

The authors of [67] compares Xen and OpenVZ in different configurations. The results showed that Xen had a large overhead, which led to an increase in OpenVZ performance. Various solutions for virtualization at the operating system level were compared to Xen in [101]. Overall, Xen achieved lower performance than container-based virtualization solutions. LXC achieved the best results among container-based solutions. However, performance isolation between virtual machines was best on Xen, which may be a drawback to OS-based virtualization solutions.

Authors of [29] present the results of a comparison of various aspects of the performance of virtualization and containerization technologies, such as I/O performance, network connection latency and computational performance. For performance testing, the authors use the following types of workload (see Tab. 3):

- PXZ, a parallel lossless data compression utility that uses LZMA algorithm;
- LINPAC, a package that provides a solution to linear equation systems using a LU factoring algorithm with partial rotation.

The authors also tested the performance of data processing on virtualized/containerized systems (see Fig. 10 to 12).

Table 3. Comparison of the performance of bare metal, docker and KVM solutions (based on [29])

Load	Bare metal	Docker	KVM without fine tuning	KVM (fine tuning)
PXZ (MB/s)	76.2 [± 0.93]	73.5 (-4 %) [± 0.64]	59.2 (-22 %) [± 1.88]	62.2 (-18 %) [± 1.33]
LINPAC (Gigaflop)	290.8 [± 1.13]	290.9 (-0 %) [± 0.98]	241.3 (-17 %) [± 1.18]	284.2 (-2 %) [± 1.45]
Random Access (GUPS)	0.0126 [± 0.00029]	0.0124 (-2 %) [± 0.00044]	0.0125 (-1 %) [± 0.00032]	Not performed

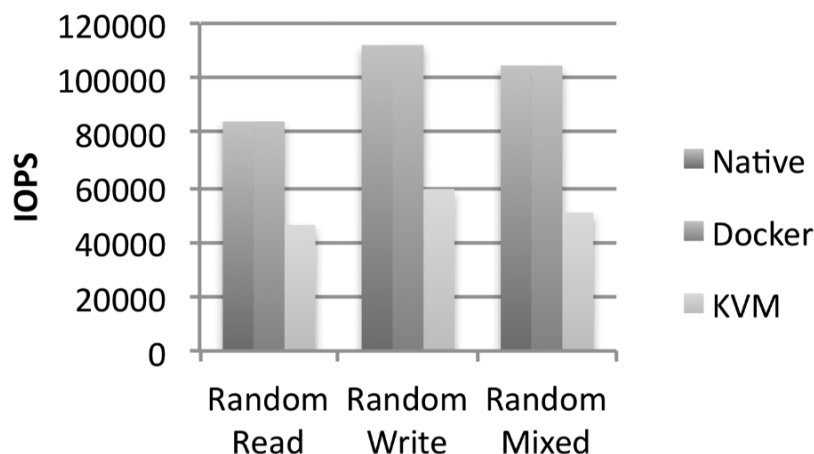


Figure 10. Testing the performance of random I/O (IOPS) (based on [29])

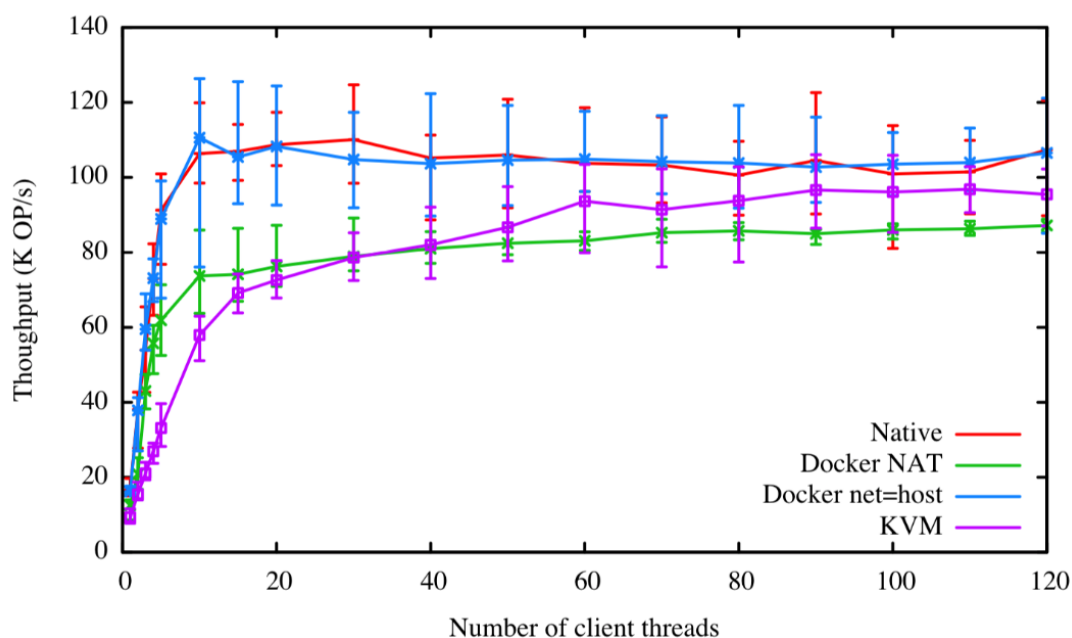


Figure 11. Performance evaluation (queries/s) of NoSQL Redis database for several deployment scenarios. Each data point is an arithmetic average obtained from 10 runs (based on [29])

The authors of [29] make the following conclusion: “In general, Docker equals or exceeds KVM performance in every case we tested... Although containers themselves have almost no overhead, Docker is not without performance gotchas. Docker volumes have noticeably better performance than files stored in AUFS. Docker’s NAT also introduces overhead for workloads with high packet rates. These features represent a tradeoff between ease of management and performance and should be considered on a case-by-case basis”.

The authors of [9] provide an analysis of the effectiveness of using virtualization and containerization technologies to solve problems in machine learning, large graphs processing and SQL queries execution (see Tab. 4) based on the Apache Spark platform. Test results show that using Spark based on Docker allows you to get acceleration more than 10 times compared to using virtual machines. However, these results vary considerably by the type of executable ap-

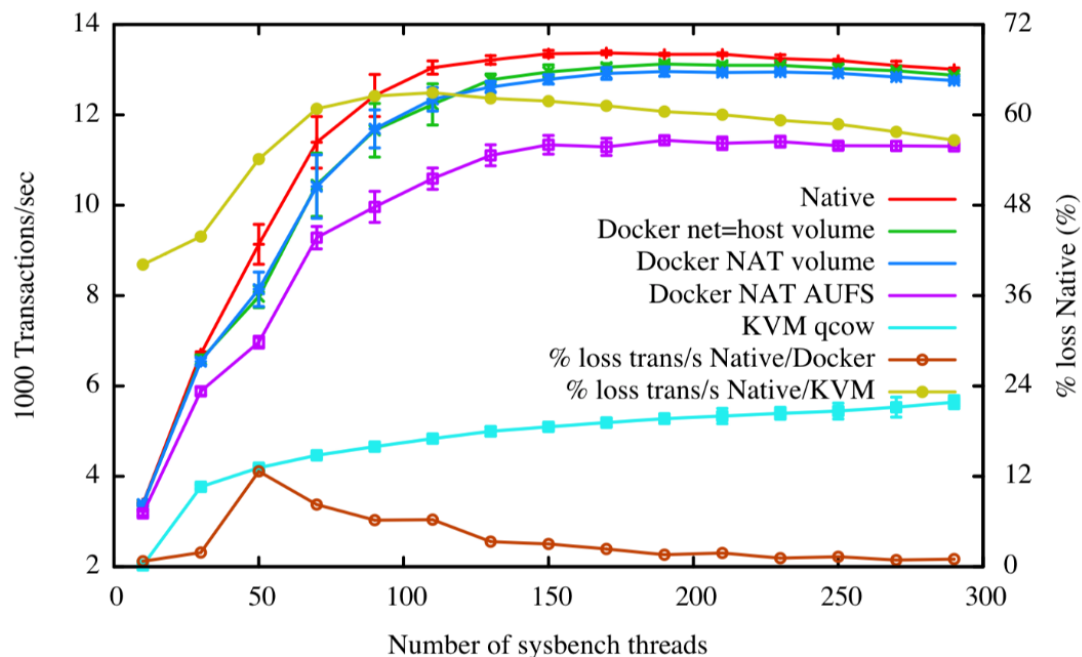


Figure 12. MySQL database performance (transactions/s) depending on parallelism (based on [29])

plications due to different load patterns and different resource management schemes (Fig. 13). In particular, it turned out that the implementation of the problem of data clustering based on the k-means method in a containerized environment is slower than inside a virtual machine. The analysis showed that this is due to a large number of mixing operations required for the implementation of the k-means algorithm, each of which causes I/O operations. The specifics of the implementation of the AUFS file system (copy-on-write technology) used in Docker, leads to the fact that a large number of I/O operations can lead to inhibition of other processes.

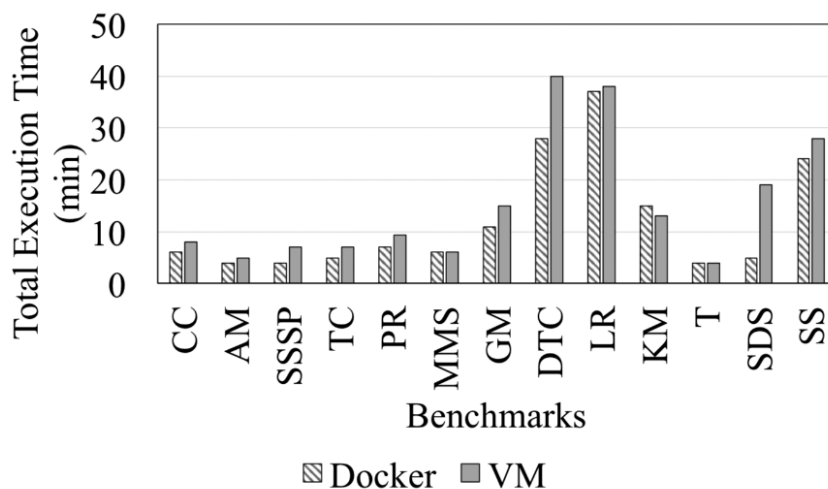


Figure 13. Comparison of the execution time of typical Spark tasks in the virtual machine environment and Docker containers (based on [9])

Table 4. List of typical data processing tasks (based on [9])

The area	Typical tasks
Machine learning	K-Means (KM) MinMaxScaler (MMS) GaussianMixture (GM) Logistic Regression (LR) DecisionTreeClassification (DTC) Tokenizer (T) Alternating Least Squares (ALS)
Graphs processing	PageRank (PR) ConnectedComponents (CC) TriangleCounting (TC) AggregateMessages (AM) Single-Source Shortest Paths (SSSP)
SQL queries	SQLDataSource (SDS) SparkSQL (SS)

As another example of usage of containerization technologies for Big Data problems, Sogand Shirinbab et al. [82] provide a performance comparison between VMware virtual machine and Docker container while running Apache Cassandra NoSQL distributed database as workload. Authors conclude that Docker had lower overhead compared to the VMware when running Cassandra.

The solution of Big Data problems often requires an effective way to utilize GPU resources. Authors of [99] conducted a comparative study of the performance of KVM, Xen and VMWare ESXi hypervisors together with LXC container management solution for execution of CUDA and OpenCL Applications. The experiments show, that LXC solution consistently performed closest to the native case. Authors also show that GPU passthrough to KVM achieves 98–100 % of the base system’s performance, while Xen and VMWare achieve 96–99 % of the base systems performance, respectively.

3.4. Conclusion

The analysis shows that virtualization and containerization technologies can be used to solve the tasks of Big Data processing as a mean of deploying specialized software platforms. Analysis of the performance of these solutions shows that, with rare exceptions, the overhead for container virtualization when deploying and executing software solutions are substantially less (from 10 % to 90 %) than for virtualization using the hypervisor. Exceptional cases are associated with the tasks that are focused on a large amount of I/O. This is due to the fact that, at the present stage of development, containers are not capable of leveling the mutual influence on each other of tasks implemented within the framework of the same computing module. In this case, containerization performance becomes comparable to the performance of virtual machine-based solutions.

4. Container Orchestration Technologies

Microservice architecture approach is aimed at solving the problem of partitioning of monolithic applications into SOA-style independently deployable services that are well supported by container architectures. Such services should be loosely coupled, supporting rapid deployment and termination.

The architectural style of microservices is an approach to developing a single application as a set of small services, each of which works in its own process and interacts with other services through standardized network protocols [83]. The life cycle of microservices should be supported by a fully automated deployment and orchestration platform. They require independent deployment and scalability depending on the current load and need, as opposed to synchronous deployment at specific times, typical for monolithic applications.

Also, the use of containerized computing services for solving real-world problems requires solving the problem of organizing their joint work. So, it is necessary to provide dependency management between containers. Orchestration plan describes components, their dependencies, and their life cycle. The PaaS cloud can then trigger workflows from the orchestration plan through agents. Software platform services can support packaging applications and their deployment from containers [56].

In this section, we discuss solutions that enable the deployment and collaboration of containerized applications within cloud infrastructures.

4.1. Private Cloud IaaS Platforms

Cloud computing is characterized by the dynamic provision of computing resources based on service level agreements between the service provider and the consumer [14]. To implement cloud computing, many open source solutions have been developed [96]. Authors of [27] present a taxonomy and architecture for cloud solutions, together with a comparison between open source cloud solutions.

The architectural and philosophical differences and similarities between the Eucalyptus [5], OpenNebula [62] cloud platforms and Nimbus [93] cloud solutions were compared in [80]. Authors of [100] compare the OpenNebula and OpenStack [63] platforms in terms of architecture, support for virtualization hypervisors, cloud APIs, and security aspects. Authors of [45] provide a comparison of the OpenStack, OpenNebula, Nimbus and Eucalyptus platforms in terms of interfaces, hypervisors, network capabilities, deployment and storage procedures, to assess the suitability of their use for the FutureGrid testing environment. The scalability of physical and virtual machine provisioning has also been verified. As a result of testing, the OpenStack platform showed the best results.

Q. Huang et al [36] compared CloudStack, Eucalyptus and OpenNebula platforms with the performance of classic server solutions. The results show that the use of cloud solutions provide about 10 % degradation in application performance due to the use of virtualization technologies. Performance deteriorates as the number of virtual machines in use increases. OpenNebula achieved better performance than other cloud solutions (CloudStack, Eucalyptus). Authors presented a comparison of the capabilities of the same cloud solutions for developing applications in the field of geophysical research in their next paper [37]. In particular, they compared their performance characteristics, including computational performance, I/O performance, memory performance, network data transfer speeds, and applications for geophysical research. The dif-

ference was observed in web application support, where OpenNebula showed slightly better performance since this system traffic is not routed through the cloud controller. OpenNebula also achieved better results in geophysical applications, albeit by a small margin as compared with the alternatives.

4.2. PaaS Clouds and Containerization

Currently, virtual machines create a fabric of the IaaS level clouds. Containers, however, look like a very suitable technology for packaging and managing applications in PaaS clouds. The PaaS model provides mechanisms for designing and deploying cloud applications, providing software infrastructure and libraries for transparently launching web services, routing requests and distributing workload between cloud resources [43].

Container platforms eliminate application deployment problems with compatible, lightweight, and virtualized packaging. Those containers, that are developed outside the PaaS platform, can be transferred to other computing environments, because the container provides encapsulation of applications. Some PaaS models are now compatible with containerization and standardized packaging of applications, for example, based on Docker. This development is part of the evolution of PaaS approach, moving towards a compatible PaaS based on containers. The first generation of PaaS solutions includes classic proprietary PaaS platforms, such as Microsoft Azure, Heroku, and Google App Engine [97].

The second generation of PaaS is built on open source solutions [7], such as Cloud Foundry [8] or OpenShift [74], which allow users to run their own PaaS (both locally and in the cloud), with integrated container support. In 2017, OpenShift switched from its own container model to the Docker model. The Cloud Foundry platform implements containerization based on its internal Diego solution [19]. Cloud Foundry and OpenShift handle containers differently. Cloud Foundry supports stateless applications through containers, but services that require state preservation are deployed in virtual machines. On the other hand, OpenShift does not distinguish them [68].

4.3. Container Clustering and Orchestration

The next problem is to facilitate the transition from one container to clusters of containers that allow running containerized applications on several clusters or in several clouds [50].

Authors of [43] proposed a general solution to ensure the formation of container clusters (Fig. 14). Within this model, each *computing cluster* consists of several nodes — virtual servers on hypervisors or, possibly, on physical servers with a single tenant. Each node contains several containers with common services that provide planning, load balancing, and application execution. Further, *application services* are logical groups of containers from the same image. Services allow you to scale applications through sites. *Volumes* are used for applications requiring persistent data. Containers can mount volumes. The data stored in these volumes is retained even after the container has been terminated. Finally, *links* allow you to create a connection and provide connectivity for two or more containers.

Deployment of distributed applications through containers is supported by a virtual scalable *head node (or head cluster)* that provides scaling, load balancing and fault tolerance of the entire system. At the same time, the API allows you to manage the life cycle of clusters. The head node of the cluster accepts commands from the outside and sends them to the container hosts. This

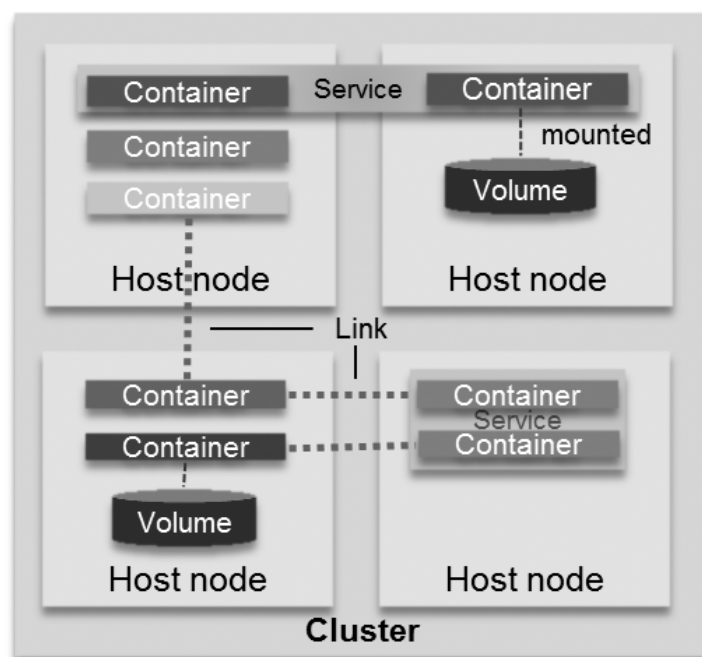


Figure 14. Containers cluster architecture (based on [43])

allows designing multi-container cloud applications without regard to the basic network topology and avoids manual configuration [32].

As part of the cluster architecture, the basic cloud infrastructure provides solutions of service discovery (for example, through shared distributed “key-value” storages), orchestration and deployment, including load balancing, monitoring, scaling, and data transfer management.

It should be noted that the OASIS organization is developing a “*Topology and Orchestration Specification for Cloud Applications*” (TOSCA) standard [11, 12, 61]. TOSCA supports several functions:

- compatible description of application cloud services and infrastructure;
- communication between parts of the service;
- operational behavior of these services (deployment, updating or shutdown) in terms of orchestration.

The implementation of such a standard provides the advantage of independence from the service supplier, as well as any particular cloud service or hosting provider. TOSCA templates can be used to define container clusters, abstract nodes and relation types, as well as application stack templates [10].

TOSCA standard defines cloud services orchestration plan on the basis of the YAML language. This orchestration plan is used to organize the deployment of applications, as well as automation processes that are implemented after deployment. The orchestration plan describes the applications and their life cycle, as well as the relationships between the components. This includes connections between applications and their locations. Using TOSCA, one can describe the service infrastructure, the intermediate computational layer of the platform, and the application layer located on the top (see Fig. 15).

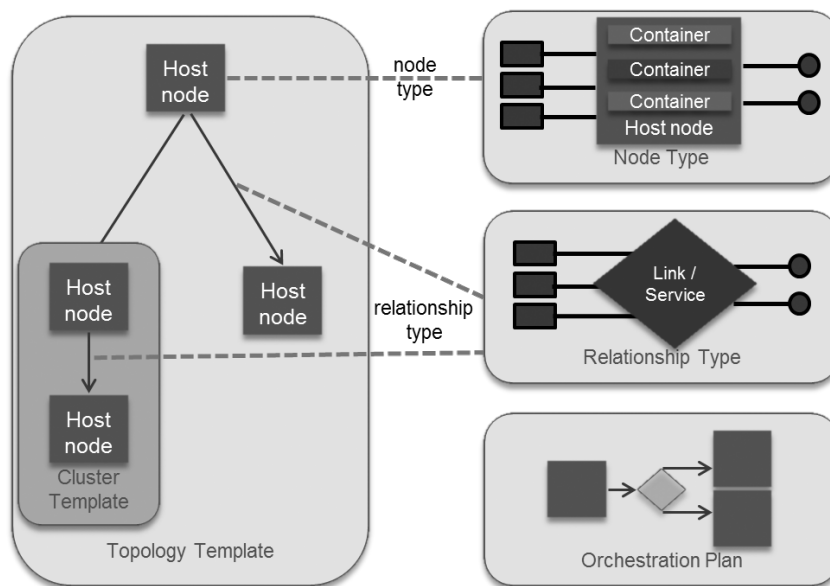


Figure 15. Reference structure for orchestrating cluster topology.

There are a number of PaaS platforms that support the TOSCA standard. For example, the Cloudify platform [20] will be able to adopt the TOSCA orchestration plan and then apply it by creating the necessary infrastructure and running the application.

4.4. Review of Container Orchestration Solutions

The goal for a container-based virtual clusters is to provide the users with computer clusters to be used as if they were physical computing clusters, with the added value of using containers instead of VMs. Therefore, the requirements for the container-based virtual cluster is to preserve the very same environment and usage patterns that are commonly used in this computing platforms, i.e. the software stack: the OS, the cluster middleware, the parallel environments and the applications, as shown in Fig. 16 [1].

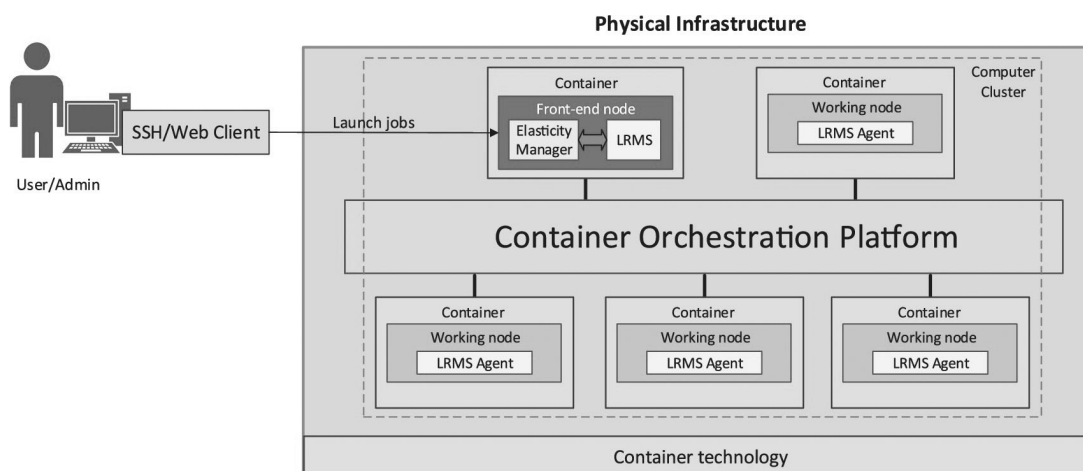


Figure 16. Generic architecture to deliver container-based virtual computer clusters deployed on a computing infrastructure managed by a Container Orchestration Platform (based on [1])

In this section, we would consider the most used systems that provide an orchestration of containerized applications. We would overview the following systems: Docker Compose, Docker Swarm, Apache Mesos, and Kubernetes.

Docker Compose is a Docker solution that provides the creation of multi-container Docker-based applications [23]. This solution is based on the YAML configuration files, which define the relationships between containers and the details of their interaction (such as images, volumes, service configurations). We can highlight the following main advantages of this project: comparative ease of implementation, convenience and ease of setup, as well as the possibility of easy distribution of cluster configurations in the format of YAML files [30]. The disadvantages include slightly less functionality compared to other projects under consideration (including, supporting high availability, etc.). Docker Compose is a suitable solution for application developers, but not functional enough to support large infrastructures.

Docker Swarm was originally a supplement to the Docker platform, but since Docker version 1.12 it has been added to the main package [24]. The Docker Swarm system provides a gradual update of services, the organization of inter-node network connections, load balancing in multi-container applications. In a Swarm, each task maps to one container. When using Docker you can control container placement decisions by using labels (tags) either user defined or system defined. The scheduler also takes into consideration CPU and memory constraints when scheduling containers (see Fig. 17) [39].

The main advantage of Docker Swarm is the built-in support for the Docker Compose configuration files. Compared to Docker Compose, this is a more advanced solution, similar to other orchestrators, although its capabilities are still limited compared to the Kubernetes ecosystem. The Docker Swarm solution is suitable for small clusters that require simple management, deployment and use in small production environments. One of the limitations of Swarm Orchestrator is its scalability, as well as the fact that it only works with Docker containers [52]. The *Apache*

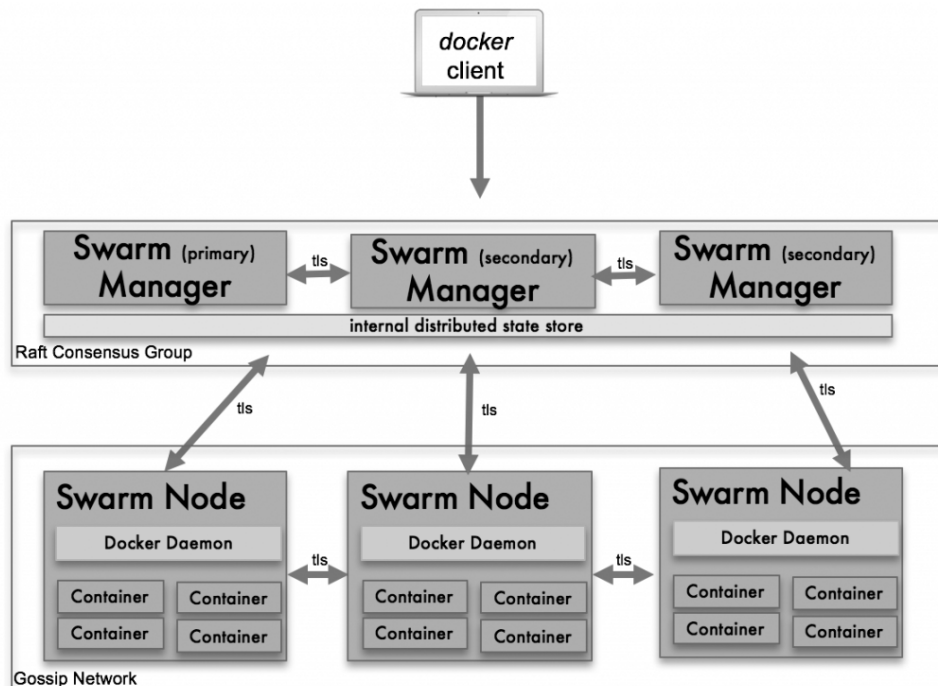


Figure 17. Docker Swarm Architecture (based on [39])

Mesos [3] project is a platform that links distributed hardware resources into a single pool of resources. These resources can be used by application frameworks to distribute the load between them. The project is based on the core of a distributed system, which follows the same principles as the Linux kernel, but at a different level of abstraction. The Mesos kernel runs on all machines in the cluster. This makes it easy to use applications with APIs for resource management and scheduling in cloud environments. The difference between Apache Mesos and other solutions is that Mesos is suitable not only for containers. This is an open source solution composed of Apache Foundation projects (such as Hadoop and Zookeeper). Container orchestration is provided by the Marathon platform [53], which is part of the Apache Mesos project. As for compatibility, it initially supports LXC, as well as Docker. This orchestration system is often used to organize the solution of issues in the field of Big Data processing [15, 52, 77, 86].

Another solution for managing clusters, albeit at a higher level than Mesos, is the *Kubernetes* system [90]. Kubernetes is an open source project developed by Google, that automates the deployment, scaling and management of container applications. This platform was built on the basis of their experience with containers over the past decade. This project is an orchestrator, which, unlike other technologies, supports several container solutions including, Docker, rkt, CRI-O, Windows containers. Kubernetes consists of two main roles: the master and the node. The master is the main component that controls each group of nodes. Nodes accept the requirements of the master and perform the actions defined by the master. Three main components are used to launch and manage containers:

- The pod is the basic unit of planning and deployment, which is a group of related containers.
- A replication controller that is a supervisor for pods and controls the status of the cluster.
- Kubernetes Services and kubelets, which run on each container and manage containers.

The Kubernetes platform is also gaining popularity in such fields as machine learning and Big Data processing. For example, authors of [46] present the BAIPAS system — a distributed platform supporting Big Data processing, artificial intelligence, and machine learning. BAIPAS uses Kubernetes and Docker to provide simple platform installation and monitoring, as well as NVIDIA Docker to provide GPU resources for deploying TensorFlow within containers.

The authors of [78] describe the computing system architecture to support the EO4Wildlife project (www.eo4wildlife.eu). In order to ensure the interaction of a wide range of researchers with a set of large geo-information data, the EO4wildlife platform was developed, providing the implementation of the Spark system based on containerized infrastructure supported by Kubernetes. In this project, Kubernetes is responsible for application deployment and management, including automatic scaling, load balancing, and monitoring of tasks.

4.5. Summary

The analysis shows that orchestration and containerization technologies are increasingly used to implement Big Data processing solutions. They can be used either explicitly, by using container orchestration systems such as Docker Compose or Kubernetes, or indirectly, by deploying applications in PaaS environments that support automatic scaling and lifecycle management of computing services, such as Cloudify or OpenShift.

The use of such systems allows one to automate the process of application deployment. It also makes it easier to repeat experiments, because files describing the computing infrastructure and the deployment of computing services can be easily distributed among researchers. Also, these technologies are well integrated with other solutions and extensions of container computing

systems, such as NVIDIA Docker, which allows them to be effectively used for solving problems related to data analysis and machine learning.

Conclusion

Solving the issues of Big Data processing is impossible without the use of distributed computing infrastructures. To date, the key technologies that support the fabric of distributed computing systems are technologies of virtualization and containerization of resources.

We have analyzed the key technologies of virtualization and containerization of computing resources used today. Virtualization has been designed to abstract hardware and system resources in order to ensure that multiple operating systems work together on the basis of one physical node. There are several approaches to the implementation of virtualization. Full virtualization is aimed at hardware emulation. Paravirtualization requires modification of the virtualized OS and coordination of operations between the virtual OS and the hypervisor. The OS-level virtualization approach (or containerization) does not require a hypervisor. Instead, the base OS is modified to ensure that several instances of the OS are able to be executed on the same machine. Although there are some disadvantages of containerization that should be addressed, especially in such cases as shared resources usage, weaker isolation, and security issues, comparing to virtual machines.

The analysis shows that virtualization and containerization technologies can be used in Big Data processing as a means of deploying of specialized software platforms. Analysis of the performance of these solutions shows that, with rare exceptions, the storage costs for container virtualization when deploying and executing software solutions are substantially less (from 10 % to 90 %) than for virtualization using the hypervisor.

Also, orchestration technologies are increasingly being used to implement Big Data processing solutions. They can be used either explicitly, by using container orchestration systems, such as Docker Compose or Kubernetes, or indirectly, by deploying applications in PaaS environments that support automatic scaling and lifecycle management of computing services, such as Cloudify or OpenShift.

The use of such systems allows one to automate the process of deployment of applications in a cloud environment. Such approach also makes it easier to repeat experiments, because files describing the computing infrastructure and the deployment of computing services can be easily distributed among researchers. Also, these technologies are well integrated with other solutions and extensions of container computing systems, such as NVIDIA Docker, which allows them to be effectively used for solving problems related to data analysis and machine learning.

It can be noted that the transition from virtualization to containerization has reduced the overhead costs associated with managing computing resources by orders of magnitude. This made it possible to efficiently use containerization technologies for solving a large class of problems requiring high performance from computing resources, including Big Data processing tasks.

Acknowledgements

This article contains the results of a project carried out as part of the implementation of the Program of the Center for Competence of the National Technology Initiative “Center for Storage and Analysis of Big Data”, supported by the Ministry of Science and Higher Education of the Russian Federation under the Contract of Moscow State University with the Fund for Support of Projects of the National Technology Initiative No. 13/1251/2018 (December 11, 2017).

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

References

1. de Alfonso, C., Calatrava, A., Moltó, G.: Container-based virtual elastic clusters. *Journal of Systems and Software* 127, 1–11 (2017), DOI: 10.1016/j.jss.2017.01.007
2. Anderson, C.: Docker. *IEEE Software* 32(3), 102–c3 (2015), DOI: 10.1109/MS.2015.62
3. Apache Software Foundation: Apache Mesos. <http://mesos.apache.org/>, accessed: 2018-12-04
4. Apache Software Foundation: Apache Tomcat. <http://tomcat.apache.org/>, accessed: 2018-11-29
5. Appscale Systems: Eucalyptus. <https://www.eucalyptus.cloud/>, accessed: 2018-11-30
6. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03*. pp. 164–177. ACM Press, New York, New York, USA (2003), DOI: 10.1145/945445.945462
7. Baset, S.A.: Open source cloud technologies. In: *Proceedings of the Third ACM Symposium on Cloud Computing - SoCC '12*. pp. 1–2. ACM Press, New York, New York, USA (2012), DOI: 10.1145/2391229.2391257
8. Bernstein, D.: Cloud Foundry Aims to Become the OpenStack of PaaS. *IEEE Cloud Computing* 1(2), 57–60 (2014), DOI: 10.1109/MCC.2014.32
9. Bhimani, J., Yang, Z., Leiser, M., Mi, N.: Accelerating big data applications using lightweight virtualization framework on enterprise cloud. In: *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. pp. 1–7. IEEE (2017), DOI: 10.1109/HPEC.2017.8091086
10. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA - A runtime for TOSCA-based cloud applications. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2013), DOI: 10.1007/978-3-642-45005-1_62
11. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications. In: *Advanced Web Services*, pp. 527–549. Springer New York, New York, NY (2014), DOI: 10.1007/978-1-4614-7535-4_22
12. Binz, T., Breiter, G., Leyman, F., Spatzier, T.: Portable Cloud Services Using TOSCA. *IEEE Internet Computing* 16(3), 80–85 (2012), DOI: 10.1109/MIC.2012.43
13. Boettiger, C.: An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review* 49(1), 71–79 (2015), DOI: 10.1145/2723872.2723882

14. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* 25(6), 599–616 (2009), DOI: 10.1016/j.future.2008.12.001
15. Canizo, M., Onieva, E., Conde, A., Charramendieta, S., Trujillo, S.: Real-time predictive maintenance for wind turbines using Big Data frameworks. In: 2017 IEEE International Conference on Prognostics and Health Management (ICPHM). pp. 70–77. IEEE (2017), DOI: 10.1109/ICPHM.2017.7998308
16. Canonical Ltd.: Linux Containers - LXD- Introduction Accessed: 2018-11-29
17. Canosa, R., Tchernykh, A., Cortés-Mendoza, J.M., Rivera-Rodriguez, R., Rizk, J.E.L., Avetisyan, A., Du, Z., Radchenko, G., Morales, E.C.: Energy consumption and quality of service optimization in containerized cloud computing. In: The Proceedings of the 2018 Ivannikov ISPRAS Open Conference (ISPRAS 2018). pp. 47–55. IEEE, Mocsow (2018), DOI: 10.1109/ISPRAS.2018.00014
18. Che, J., Shi, C., Yu, Y., Lin, W.: A Synthetical Performance Evaluation of OpenVZ, Xen and KVM. In: 2010 IEEE Asia-Pacific Services Computing Conference. pp. 587–594. IEEE (2010), DOI: 10.1109/APSCC.2010.83
19. Cloud Foundry Foundation: Diego Components and Architecture | Cloud Foundry Docs. <https://docs.cloudfoundry.org/concepts/diego/diego-architecture.html>, accessed: 2018-12-04
20. Cloudify Platform: Cloud NFV Orchestration Based on TOSCA | Cloudify. <https://cloudify.co/>, accessed: 2018-12-05
21. CRI-O author: Cri-o. <http://cri-o.io/>, accessed: 2018-11-30
22. Docker Inc.: Docker - Build, Ship, and Run Any App, Anywhere. <https://www.docker.com/>, accessed: 2019-02-28
23. Docker Inc.: Docker Compose | Docker Documentation. <https://docs.docker.com/compose/>, accessed: 2018-12-04
24. Docker Inc.: Swarm mode key concepts | Docker Documentation. <https://docs.docker.com/engine/swarm/key-concepts/>, accessed: 2018-12-04
25. Dua, R., Raja, A.R., Kakadia, D.: Virtualization vs Containerization to Support PaaS. In: 2014 IEEE International Conference on Cloud Engineering. pp. 610–614. IEEE (2014), DOI: 10.1109/IC2E.2014.41
26. Duato, J., Pena, A.J., Silla, F., Mayo, R., Quintana-Orti, E.S.: rCUDA: Reducing the number of GPU-based accelerators in high performance clusters. In: 2010 International Conference on High Performance Computing & Simulation. pp. 224–231. IEEE (2010), DOI: 10.1109/HPCS.2010.5547126
27. Dukaric, R., Juric, M.B.: Towards a unified taxonomy and architecture of cloud frameworks. *Future Generation Computer Systems* 29(5), 1196–1210 (2013), DOI: 10.1016/j.future.2012.09.006

28. Fayyad-Kazan, H., Perneel, L., Timmerman, M.: Full and Para-Virtualization with Xen: A Performance Comparison. *Journal of Emerging Trends in Computing and Information Sciences* 4(9), 719–727 (2013)
29. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and Linux containers. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). pp. 171–172. IEEE (2015), DOI: 10.1109/ISPASS.2015.7095802
30. Filgueira, R., Da Silva, R.F., Deelman, E., Christodoulou, V., Krause, A.: IoT-Hub: New IoT data-platform for Virtual Research Environments. In: 10th International Workshop on Science Gateways (IWSG 2018). pp. 13–15 (2018)
31. Firesmith, D.: Virtualization via Containers. https://insights.sei.cmu.edu/sei_blog/2017/09/virtualization-via-containers.html (2017), accessed: 2018-02-28
32. Gass, O., Meth, H., Maedche, A.: PaaS Characteristics for Productive Software Development: An Evaluation Framework. *IEEE Internet Computing* 18(1), 56–64 (2014), DOI: 10.1109/MIC.2014.12
33. Gerdau, B.L., Weier, M., Hinkenjann, A.: Containerized Distributed Rendering for Interactive Environments. In: EuroVR 2017: Virtual Reality and Augmented Reality. pp. 69–86 (2017), DOI: 10.1007/978-3-319-72323-5_5
34. Giunta, G., Montella, R., Agrillo, G., Coviello, G.: A GPGPU Transparent Virtualization Component for High Performance Computing Clouds. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 379–391 (2010), DOI: 10.1007/978-3-642-15277-1_37
35. Gupta, V., Gavrilovska, A., Schwan, K., Kharche, H., Tolia, N., Talwar, V., Ranganathan, P.: GViM: GPU-accelerated virtual machines. In: *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing - HPCVirt '09*. pp. 17–24. ACM Press, New York, New York, USA (2009), DOI: 10.1145/1519138.1519141
36. Huang, Q., Xia, J., Yang, C., Liu, K., Li, J., Gui, Z., Hassan, M., Chen, S.: An experimental study of open-source cloud platforms for dust storm forecasting. In: *Proceedings of the 20th International Conference on Advances in Geographic Information Systems - SIGSPATIAL '12*. p. 534. ACM Press, New York, New York, USA (2012), DOI: 10.1145/2424321.2424408
37. Huang, Q., Yang, C., Liu, K., Xia, J., Xu, C., Li, J., Gui, Z., Sun, M., Li, Z.: Evaluating open-source cloud computing solutions for geosciences. *Computers & Geosciences* 59, 41–52 (2013), DOI: 10.1016/j.cageo.2013.05.001
38. Huber, N., von Quast, M., Hauck, M., Kounev, S.: Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments. In: *Proceedings of the 1st International Conference on Cloud Computing and Services Science*. pp. 563–573. SciTePress - Science and and Technology Publications (2011), DOI: 10.5220/0003388905630573

39. IT Solution Architects: Containers 102: Continuing the Journey from OS Virtualization to Workload Virtualization. <https://medium.com/@ITsolutions/containers-102-continuing-the-journey-from-os-virtualization-to-workload-virtualization-54fe5576969d> (2017), accessed: 2019-02-27
40. Kang, D., Jun, T.J., Kim, D., Kim, J., Kim, D.: ConVGPU: GPU Management Middleware in Container Based Virtualized Environment. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER). pp. 301–309. IEEE (2017), DOI: 10.1109/CLUSTER.2017.17
41. Kim, J., Jun, T.J., Kang, D., Kim, D., Kim, D.: GPU Enabled Serverless Computing Framework. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP). pp. 533–540. IEEE (2018), DOI: 10.1109/PDP2018.2018.00090
42. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: kvm: the Linux Virtual Machine Monitor. In: Ottawa Linux Symposium (2007), DOI: 10.1186/gb-2008-9-1-r8
43. Koukis, V., Venetsanopoulos, C., Koziris, N.: ~okeanos: Building a Cloud, Cluster by Cluster. IEEE Internet Computing 17(3), 67–71 (2013), DOI: 10.1109/MIC.2013.43
44. KVM contributors: Kernel Virtual Machine. https://www.linux-kvm.org/page/Main_Page, accessed: 2018-11-29
45. von Laszewski, G., Diaz, J., Wang, F., Fox, G.C.: Comparison of Multiple Cloud Frameworks. In: 2012 IEEE Fifth International Conference on Cloud Computing. pp. 734–741. IEEE (2012), DOI: 10.1109/CLOUD.2012.104
46. Lee, M., Shin, S., Hong, S., Song, S.k.: BAIPAS: Distributed Deep Learning Platform with Data Locality and Shuffling. In: 2017 European Conference on Electrical Engineering and Computer Science (EECS). pp. 5–8. IEEE (2017), DOI: 10.1109/EECS.2017.10
47. Li, J., Wang, Q., Jayasinghe, D., Park, J., Zhu, T., Pu, C.: Performance Overhead among Three Hypervisors: An Experimental Study Using Hadoop Benchmarks. In: 2013 IEEE International Congress on Big Data. pp. 9–16. IEEE (2013), DOI: 10.1109/Big-Data.Congress.2013.11
48. Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., Smith, S., Hand, S., Crowcroft, J.: Unikernels. ACM SIGPLAN Notices 48(4), 461–472 (2013), DOI: 10.1145/2499368.2451167
49. Madhavapeddy, A., Scott, D.J.: Unikernels. Communications of the ACM 57(1), 61–69 (2014), DOI: 10.1145/2541883.2541895
50. Martin-Flatin, J.: Challenges in Cloud Management. IEEE Cloud Computing 1(1), 66–70 (2014), DOI: 10.1109/MCC.2014.4
51. Mavridis, I., Karatza, H.: Performance and Overhead Study of Containers Running on Top of Virtual Machines. In: 2017 IEEE 19th Conference on Business Informatics (CBI). pp. 32–38. IEEE (2017), DOI: 10.1109/CBI.2017.69

52. Mercl, L., Pavlik, J.: The Comparison of Container Orchestrators. In: Third International Congress on Information and Communication Technology, pp. 677–685. Springer (2019), DOI: 10.1007/978-981-13-1165-9_62
53. Mesosphere Inc.: Marathon: A container orchestration platform for Mesos and DC/OS. <https://mesosphere.github.io/marathon/>, accessed: 2018-12-05
54. Microsoft: Windows Containers on Windows Server | Microsoft Docs. <https://docs.microsoft.com/en-us/virtualization/windowscontainers/quick-start/quick-start-windows-server>, accessed: 2018-11-29
55. Naik, N.: Migrating from Virtualization to Dockerization in the Cloud: Simulation and Evaluation of Distributed Systems. In: 2016 IEEE 10th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA). pp. 1–8. IEEE (2016), DOI: 10.1109/MESOCA.2016.9
56. Noor, T.H., Sheng, Q.Z., Ngu, A.H., Dustdar, S.: Analysis of Web-Scale Cloud Services. *IEEE Internet Computing* 18(4), 55–61 (2014), DOI: 10.1109/MIC.2014.64
57. NVIDIA: GPU-Enabled Docker Container | NVIDIA. <https://www.nvidia.com/object/docker-container.html>, accessed: 2018-12-01
58. NVIDIA: NVIDIA Container Runtime | NVIDIA Developer. <https://developer.nvidia.com/nvidia-container-runtime>, accessed: 2018-11-30
59. NVIDIA: nvidia-docker. <https://github.com/NVIDIA/nvidia-docker/wiki/nvidia-docker>, accessed: 2018-11-29
60. NVIDIA: nvidia docker plugin. <https://github.com/NVIDIA/nvidia-docker/wiki/nvidia-docker-plugin>, accessed: 2018-11-30
61. OASIS: OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca, accessed: 2018-11-28
62. OpenNebula Project: Home - OpenNebula. <https://openebula.org/>, accessed: 2018-12-04
63. OpenStack Foundation: OpenStack: Build the future of Open Infrastructure. <https://www.openstack.org/>, accessed: 2018-12-04
64. OpenVZ community: Open source container-based virtualization for Linux. <https://openvz.org/>, accessed: 2018-11-29
65. Oracle: Changelog for VirtualBox 4.1. <https://www.virtualbox.org/wiki/Changelog-4.1>, accessed: 2019-06-03
66. Oracle: VirtualBox. <https://www.virtualbox.org/>, accessed: 2018-11-29
67. Padala, P., Zhu, X., Wang, Z., Singhal, S., Shin, K.G.: Performance Evaluation of Virtualization Technologies for Server Consolidation. HP Technical Reports (2007), DOI: 10.1.1.70.4605

68. Pahl, C., Lee, B.: Containers and clusters for edge cloud architectures-A technology review. In: Proceedings - 2015 International Conference on Future Internet of Things and Cloud, FiCloud 2015 and 2015 International Conference on Open and Big Data, OBD 2015 (2015), DOI: 10.1109/FiCloud.2015.35
69. Palit, H.N., Li, X., Lu, S., Larsen, L.C., Setia, J.A.: Evaluating hardware-assisted virtualization for deploying HPC-as-a-service. In: Proceedings of the 7th international workshop on Virtualization technologies in distributed computing - VTDC '13. p. 11. ACM Press, New York, New York, USA (2013), DOI: 10.1145/2465829.2465833
70. Pasztor, J.: LXC vs Docker. <https://pasztor.at/blog/lxc-vs-docker> (2018), accessed: 2018-02-28
71. Pivotal Software Inc.: GETTING STARTED: Building an Application with Spring Boot. <https://spring.io/guides/gs/spring-boot/>, accessed: 2018-11-29
72. Qanbari, S., Li, F., Dustdar, S.: Toward Portable Cloud Manufacturing Services. IEEE Internet Computing 18(6), 77–80 (2014), DOI: 10.1109/MIC.2014.125
73. Ranjan, R.: The Cloud Interoperability Challenge. IEEE Cloud Computing 1(2), 20–24 (2014), DOI: 10.1109/MCC.2014.41
74. Red Hat Inc.: OpenShift: Container Application Platform by Red Hat, Built on Docker and Kubernetes. <https://www.openshift.com/>, accessed: 2018-12-04
75. Red Hat Inc.: rkt, a security-minded, standards-based container engine. <https://coreos.com/rkt/>, accessed: 2018-11-30
76. Red Hat Inc.: WildFly. <http://wildfly.org/>, accessed: 2018-11-29
77. Reuther, A., Byun, C., Arcand, W., Bestor, D., Bergeron, B., Hubbell, M., Jones, M., Michaleas, P., Prout, A., Rosa, A., Kepner, J.: Scheduler technologies in support of high performance data analysis. In: 2016 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–6. IEEE (2016), DOI: 10.1109/HPEC.2016.7761604
78. Sabeur, Z.A., Correndo, G., Veres, G., Arbab-Zavar, B., Lorenzo, J., Habib, T., Haugomard, A., Martin, F., Zigna, J.M., Weller, G.: EO Big Data Connectors and Analytics for Understanding the Effects of Climate Change on Migratory Trends of Marine Wildlife. In: ISESS 2017: Environmental Software Systems. Computer Science for Environmental Protection. pp. 85–94. Zadar (2017), DOI: 10.1007/978-3-319-89935-0_8
79. Sarai, A.: CVE-2019-5736: runc container breakout (all versions). <https://seclists.org/oss-sec/2019/q1/119>, accessed: 2019-03-07
80. Sempolinski, P., Thain, D.: A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In: 2010 IEEE Second International Conference on Cloud Computing Technology and Science. pp. 417–426. IEEE (2010), DOI: 10.1109/CloudCom.2010.42
81. Shi, L., Chen, H., Sun, J., Li, K.: vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines. IEEE Transactions on Computers 61(6), 804–816 (2012), DOI: 10.1109/TC.2011.112

82. Shirinbab, S., Lundberg, L., Casalicchio, E.: Performance evaluation of container and virtual machine running cassandra workload. In: 2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech). pp. 1–8. IEEE (2017), DOI: 10.1109/CloudTech.2017.8284700
83. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop distributed file system. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010 (2010), DOI: 10.1109/MSST.2010.5496972
84. Singh, R.: LXD vs Docker. <https://linuxhint.com/lxd-vs-docker/>, accessed: 2018-11-29
85. Soltesz, S., Pötzl, H., Fiuczynski, M.E., Bavier, A., Peterson, L.: Container-based operating system virtualization. ACM SIGOPS Operating Systems Review 41(3), 275 (2007), DOI: 10.1145/1272998.1273025
86. Soualhia, M., Khomh, F., Tahar, S.: Task Scheduling in Big Data Platforms: A Systematic Literature Review. Journal of Systems and Software 134, 170–189 (2017), DOI: 10.1016/j.jss.2017.09.001
87. SUSE LLC: SUSE Linux Enterprise Server 11 SP4 Virtualization with KVM. https://www.suse.com/documentation/sles11/singlehtml/book_kvm/book_kvm.html#cha.kvm.limits, accessed: 2019-03-06
88. The Eclipse Foundation: Eclipse Jetty. <https://www.eclipse.org/jetty/>, accessed: 2018-11-29
89. The Linux Foundation: Home - Open Containers Initiative. <https://www.opencontainers.org/>, accessed: 2018-11-29
90. The Linux Foundation: Production-Grade Container Orchestration - Kubernetes. <https://kubernetes.io/>, accessed: 2018-12-04
91. The Linux Foundation: Xen Project Release Features. https://wiki.xen.org/wiki/Xen_Project_Release_Features#Limits, accessed: 2019-03-06
92. Uhlig, R., Neiger, G., Rodgers, D., Santoni, A., Martins, F., Anderson, A., Bennett, S., Kagi, A., Leung, F., Smith, L.: Intel virtualization technology. Computer 38(5), 48–56 (2005), DOI: 10.1109/MC.2005.163
93. University of Chicago: Nimbus. <http://www.nimbusproject.org/>, accessed: 2018-12-04
94. VMware: VMware Configuration Maximus. <https://configmax.vmware.com/>, accessed: 2019-03-06
95. VMware: VMware ESXi: The Purpose-Built Bare Metal Hypervisor. <https://www.vmware.com/products/esxi-and-esx.html>, accessed: 2018-11-29
96. Voras, I., Mihaljević, B., Orlić, M., Pletikosa, M., Žagar, M., Pavić, T., Zimmer, K., Čavrak, I., Paunović, V., Bosnić, I., Tomić, S.: Evaluating open-source cloud computing solutions. In: Proceedings of the 34th International Convention for Information and Communication Technology, Electronics and Microelectronics (MIPRO 2011). pp. 209–214. Opatija (2011)

97. Walraven, S., Truyen, E., Joosen, W.: Comparing PaaS offerings in light of SaaS development. *Computing* 96(8), 669–724 (2014), DOI: 10.1007/s00607-013-0346-9
98. Walters, J.P., Chaudhary, V., Cha, M., Jr., S.G., Gallo, S.: A Comparison of Virtualization Technologies for HPC. In: 22nd International Conference on Advanced Information Networking and Applications (aina 2008). pp. 861–868. IEEE (2008), DOI: 10.1109/AINA.2008.45
99. Walters, J.P., Younge, A.J., Kang, D.I., Yao, K.T., Kang, M., Crago, S.P., Fox, G.C.: GPU Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications. In: 2014 IEEE 7th International Conference on Cloud Computing. pp. 636–643. IEEE (2014), DOI: 10.1109/CLOUD.2014.90
100. Wen, X., Gu, G., Li, Q., Gao, Y., Zhang, X.: Comparison of open-source cloud management platforms: OpenStack and OpenNebula. In: 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery. pp. 2457–2461. IEEE (2012), DOI: 10.1109/FSKD.2012.6234218
101. Xavier, M.G., Neves, M.V., Rossi, F.D., Ferreto, T.C., Lange, T., De Rose, C.A.F.: Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. In: 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. pp. 233–240. IEEE (2013), DOI: 10.1109/PDP.2013.41
102. Xen Project community: Xen Project wiki: Dom0. <https://wiki.xenproject.org/wiki/Dom0>, accessed: 2018-11-29
103. Younge, A.J., Henschel, R., Brown, J.T., von Laszewski, G., Qiu, J., Fox, G.C.: Analysis of Virtualization Technologies for High Performance Computing Environments. In: 2011 IEEE 4th International Conference on Cloud Computing. pp. 9–16. IEEE (2011), DOI: 10.1109/CLOUD.2011.29
104. Zhenyun Zhuang, Cuong Tran, Weng, J., Ramachandra, H., Sridharan, B.: Taming memory related performance pitfalls in linux Cgroups. In: 2017 International Conference on Computing, Networking and Communications (ICNC). pp. 531–535. IEEE (2017), DOI: 10.1109/IC-CNC.2017.7876184