Further investigation of the performance overhead for hypervisor- and container-based virtualization.

By

Cui Zheng

Supervisor: Maria Kihl

Examiner: Christian Nyberg

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University, Sweden

Abstract

This thesis work aims to find a suitable environment for different virtualization system to achieve enhanced performance. Virtualization plays a vital role in cloud services, and it is essential to help cloud users comprehend the brief distinction between different virtualization technologies. The hypervisor-based technology has been used as the primary selection for cloud services in the past, but container-based virtualization starts receiving more attention and is regarded as the substitution of hypervisor-based technology. This thesis, contains research and discussion of the performance overhead and performance variability overhead that a Virtual Machine or Docker container introduce under certain conditions. The challenge was to find a feasible solution to measure the parameters based on the performance, acting on the CPU, memory, hardware disk, and network throughput on the physical machine. These parameters will be used in the performance comparison of these two virtualization techniques.

Moreover, in the 5G network, network function virtualization (NFV) is an efficient solution against the massive traffic, but it suffers from software-based solutions. The main concept in NFV is to decouple the network functions of the dedicated hardware working only for the given purpose. When setting up a virtualization proxy or deploying a web server within a virtual environment, it is necessary to determine if the virtualization method should use hypervisor-based (KVM) or container-based (Docker) technology. Therefore, we need to investigate as a special condition which technique will maximize the network data throughput and minimize the physical machine's overhead.

Our experiment presents the performance overhead when a webserver is running in the KVM and Docker virtualization environment. In most cases, the Docker virtualization performs better. KVM has similar performance overhead as Docker when the network is not under heavy load.

Popular science summary

Virtualization was developed by IBM in 1960[20], and it becomes a fundamental part of clouding technology these days. In computing technology, virtual operating system and virtual network resource are widely used. The first virtual machine was designed by IBM in 1967[14], those VMs are created and managed by a Hypervisor. The main function of the hypervisor is enabled multiple operating system running on the same underlying hardware. The severs deployed by the Hypervisor is known as the Host machine and those VMs running on top of the host machine is named guest machine. Recently there is a new technology implemented in the cloud named Containers. The containers are more lightweight compared with the hypervisor. The containers access the same operating system which including the system root files, libraries and common files, and they can run multiple isolated processes on the same host. Also, containers provide isolating the user space instances while sharing the same kernel; it uses the control groups and namespace technologies to provide the resource management.

The virtualization commonly has an impact on the real-world application such as network virtualization. Network function virtualization is a part of the software-defined network, which can integrate the network resource flexible with the virtual CPUs or Memory to any other VNFs instance. The network management layer can modify the data throughput concerning the performance of the network and scale expectations over a single virtualized system or operating platform. It is an efficient solution against the massive data traffic problem in 5G network.

Acknowledgement

I would like to express my gratitude to everyone who helped me to finish this thesis work

First, I would like to express my great appreciation to my supervisor Maria Kihl, who has given me lots of review and feedback to support me through this thesis.

Also, I would like to thank to my friend and family for all their best wishes.

Cui Zheng

List of Acronyms

VM	Virtual Machine
NFV	Network Function Virtualization
VNF	Virtualized Network Functions
KVM	Kernel-based Virtualization machine
LXC	Linux Container
COW	Copy-on-write
EPC	Evolved Packet Core
HA	High availability
RDD	Resilient Distributed Datasets
VBD	Virtual block device
UFW	Uncomplicated firewall

Table of contents

Abstract 3
Popular science summary 5
Acknowledgement 7
List of Acronyms 9
Table of contents 11
1. Introduction 13
2. Related work 15
3. Theory 21
3.1 Hypervisor-based virtualization 21
3.1.1. KVM 21
3.1.2. File system journaling 22
3.1.3. Performance on ARM architecture 24
3.1.4. Xen 24
3.2 Container-based virtualization 25
3.2.1 Linux containers (LXC) 26
3.2.2 Containers executed on top of VM 28
3.2.3 Docker 29
3.2.4 Memory resource management 31
3.2.5 Container performance comparison in NFV 32
3.3 Network Function Virtualization and Software Defined Network 33
3.3.1 Virtualization Comparison under Open5GCore 33
3.3.2 Network traffic simulation with Open5GMTC 34
3.4 Benchmark Application 36
3.4.1 System performance 36

3.4.2 CPU performance 36
3.4.3 Memory performance 37
3.4.3 Disk I/O performance 37
3.4.4 Network throughput 38
4. Experimental part 39
4.1 Testbed 39
4.1.1 Host machines deployment 40
4.2 Experiments 43
4.2.1 System's resource consumption 44
4.2.2 Elapsed time of the response 44
5. Results 45
5.1 Light Traffic 45
5.2 Medium Traffic 45
5.3 Heavy traffic 46
5.4 Average response time 47
6. Conclusions 49
References 52

1. Introduction

Virtualization plays a vital role in cloud services. In cloud computing, there are two kinds of virtualization technologies, hypervisor-based and container-based virtualization. A cloud user should understand the characteristic of different virtualization technologies so that they can choose the most suitable one to deploy for their services, as various cloud providers such as Amazon EC2, Google Compute Engine and Microsoft Azure use different virtualization technology.

In 1960, IBM introduced a technology called virtualization [14], and they developed the first VM in 1967. They created the Hypervisor, which is a software to manage running multiple operating systems on the defined hardware. The hypervisor virtualization deployed on the host machine is controlled and connected to the virtual machine that runs on top of it. By virtualizing system resources such as CPUs, Memory etc., it can provide an environment running multiple operating systems. Recently, containers have been deployed in cloud infrastructure. The containers access the same operating system including the system root files, libraries and common files, and they can run multiple isolated processes on the same host. Also, containers provide isolated user space instances while sharing the same kernel; it uses the control groups and namespace technologies to provide the resource management.

Generally, VMs have outstanding performance in isolation compared with containers, such as preventing VMs from interfering with each other. The currently existing virtualization technologies are hypervisor-based and container-based virtualization, which respectively corresponds to hardware-based and operating system-based virtualization method.

In this thesis, the performance of hypervisor and container-based technologies are compared within the same operating environment. Several simulations have been implemented based on these two virtualization technologies. The performance overheads of the virtualization technologies could vary not only on a feature basis, but also on task basis. With the help of a micro-benchmark tool, we evaluate the performance related to the CPU consumption and network throughput in different virtualization environments. It is shown that

container-based virtualization performs better when the web server dealing with the network request is deployed on top of it.

For further investigation, the virtual environment for data storage purposes that is deployed with the system resource virtualization related to the Disk I/O can be regarded as one of the research directions. Also, the memory utilization rate is an exciting topic when the virtualization handles the intermediate data that frequently access the system memory. At last, combined hypervisor and container technologies may become one attractive topics. Both the lightweight deployment and the isolation may be achieved when the container is running on top of the KVM.

2. Related work

There are several studies that have analyzed the performance overhead and variability overhead for the two technologies.

Zheng Li et al. [1] presented a performance comparison of the hypervisor and container-based virtualization. They used a local machine connecting to a cloud service provider (Amazon EC2A) which runs the individual benchmarks on three types of resources (physical machine, container and virtual machine) independently. The testing stage contains four parts, which respectively are communication, computing, memory, and storage. The paper evaluated with three types of resources to support the Cloud service. They noticed that the container's average performance is generally better than VM's and even comparable to the physical machine regarding many features. However, there are still some cases where VMs have better performance than containers, for example, when solving the N-Queens problem or writing small-size data to the disk.

Further, Ericsson Research, Nomadic Lab had an article [2] published, which contained a comparison between the hypervisor and lightweight virtualization. They used the Linux system with KVM as an example of a hypervisor-based system, and compared this with Docker and LXC, represented as container-based solutions. They used benchmark tools to measure CPU, memory, disk I/O and network I/O performance. They conclude that the container-based solution is more lightweight, thus facilitating the denser deployment of services. Their results show that the KVM hypervisor performance improved but still lack performance in DISK I/O efficiency. The containers' overhead could be considered almost negligible. By analyzing the measurement results, they conclude that the versatility and ease management, which are the intrinsic attribute of the containers are against the security.

A further work is relating to "Linux Container Daemon" [3]", which is a new type of container that can claim to offer improved support for security without losing the performance benefits mentioned in the end.

Prof. Ann Mary Joy [8] presented a performance comparison between Linux Containers and Virtual Machines. Several features about the container benefits listed in the article include portable deployment, fast application delivery, scale and deploy with ease and higher workloads with greater density. By making the application

performance comparison and scalability comparison, the author concludes that containers have outperformed virtual machines in regarding performance and scalability. With better scalability and resource utilization, containers have an advantage over the reduced resource overhead. However, there is one case where virtual machines overcome the containers, as mentioned by the author. This case concerns applications running with business-critical data, and here virtual machines are better, since the containers run root privileges and this may cause security issues.

When a user considers deploying their service to the cloud, it is essential to understand the performance of different virtualization technologies when choosing from the cloud provider, in order to avoid degradation of the quality of service. Some papers found better performance for some of the application due to optimizations in the hypervisor.

Sampath Kumar [6] did several benchmarks measuring the performance on CPU, memory and disk I/O for Xen, LXC and KVM. He concluded that LXC is preferred when virtualizing infrastructure that is dynamic by designing for applications with secure resource isolation. KVM did a better job when memory needs more frequently access. Meanwhile, Xen performs better when disk access has a distinct signature.

Bo Wang presented [7] a performance comparison between hypervisor and container-based virtualization for the cloud user. Two macro benchmarks were used in the data analysis, HPL for high-performance computing applications and YCSB for online transaction processing applications. He concluded that for network I/O, computing rate and bandwidth of memory, both virtualizations have small overheads. Due to the cost of the hypervisor, Xen has higher overheads on operating system latencies, main memory accesses and disk, network I/O. Then they deployed an HPC application for the data analysis, and showed that the performance degradation by these two virtualization solutions is negligible, due to the small performance overhead on CPU and memory bandwidth. After that, OLTP applications were deployed to determine the transaction latency under specific loads. Here, Xen has higher overhead than Docker because of the performance overhead on the latencies of system operations, and I/Os. However, when considering the transaction throughput, Xen did

a better job due to improvements to I/O, virtual block device (VBD) [5] and aggressively data prefetching, which do not affect disk I/Os for large blocks.

Further, these technologies apply to NFV. NFV is regarded as one of the efficient solutions for the massive traffic in the 5G network, and decouples the network function of the local machine and replaces it by commodity hardware.

The author Sun Young Chung [10] investigated NFV performance in the cellular network. The paper applied KVM and Docker to virtualized SOCKS proxies and deployed it to enable MPTCP connection. By comparing latency and data throughput, we can have a better understanding when comparing these two technologies under the NFV environment. Two conclusions are listed separately. The first conclusion relates to the usage of the MPTCP connections and show that MPTCP has better performance in long and large volumes of TCP connections, and the latency between the proxy and server determines the throughput. If the latency is large (RTT 100ms), the MPTCP performs worse than the single TCP connections. The second conclusion is that while using these two technologies deployed on NFV, Docker handles the traffic with less resource consumption compare with KVM.

It is difficult to deploy an NFV system since when configuring fault management policies, multiple possibilities exist, so selecting suitable virtualization technologies and management products also needs to be considered.

Domenico Cotroneo presented an article [13] where he analyzed the dependability benchmark for NFV systems, and concluded the characteristic of container-based and hypervisor-based technologies. The paper points out that even if containers can achieve higher performance and manageability, they still perform less dependable compared with hypervisors. VMware configuration showed a higher fault detection coverage, due to a more complex fault management mechanism. In the NFV system, Docker has a memory overload problem that is unreported to the operating system. Also, Docker has some specific actions like internal kernel errors, and I/O errors that must configure the recovery actions by forcing a reboot to trigger the fault management process. Those points of view support the selection of virtualization technologies deployed in a different environment.

When deploying cloud servers for commercial use or emergency use, a high availability (HA) must be considered. When virtualization vendors offer solutions to the customers, high availability is guaranteed by setting multiple levels of failover capacity in the system. Moreover, HA refers to the measurement of the ability of the system.

Wubin Li presented an article [11] about two types of technologies for virtualized platform achieving high availability. Some main features achieving HA are live migration, VM monitoring, failure detection and check-point restore. These features are mainly for hypervisor-based technologies. For container-based technologies, because of the strong isolation features, the capability of process checkpoint /restore is essential. For LXC/Docker, this is the only possible way to achieve high availability of the system.

There is an exception in container-based technologies named OpenZC [20] that can manage live migration and checkpoint/restore by implementing loadable kernel modules plus a set of user-space utilities, while using the file system change tracking, lazy migration and interactive migration. In [20] it is concluded that there are no mature features for continuous monitoring to detect a failure of a container and automatic operation failover actions. Therefore, extensions on top of container technologies are necessary for an HA perspective.

Today, it is common to use a lightweight virtualization framework on an enterprise cloud to accelerate a big data application.

Jabki Bhimani [12] presented a performance comparison of different Apache Spark applications using both VM and Docker containers. They studied execution latency and resource utilization, which include CPU, disk, memory, etc. Spark as a new framework caches all intermediate dates (Resilient Distributed Datasets (RDDs)) in memory instead of disk. RDDs can store into memory without requiring replication and disk access. Hypervisor-based virtualization is widely used for the implementation of Spark, but recently Docker has been concerned. Both technologies can achieve resource isolation, but they have their own way to do resource management. Containers perform shared resource management while VMs perform distributed resource management. In the paper, they studied the execution time when Spark applications operated either on VM or on Docker. The container performs mostly better than VMs because of the fast setup time and

dominates read operations compared with VMs. The read operation benefits from Docker's storage driver and performs Copy-on-Write (COW).

However, there are still some cases where VMs perform better than containers, for example, K (clusters)-means algorithm. K-means has characteristics that include shuffle and intensive. The Docker-AFUS (Advanced multi-layered unification filesystem) system executed COW for every writes operation, and during the shuffle, many COW operations were stacked, which may result in the throttling stall of operating threads. Therefore, for the intensive shuffle application, VM performs better than Docker. For resource utilization, Docker has higher CPU and disk utilization ratios. Docker has lower memory utilization compared with VMs due to the bypasses of the guest OS, so it required less memory usage.

3. Theory

3.1 Hypervisor-based virtualization

One of the most important characteristics for the hypervisor is hardware-based virtualization. A virtual machine is created and managed by the hypervisor, and this enables running multiple operating systems in the same underlying hardware. The host server machine is implemented by the hypervisor, which controls and connect with the guest machine that runs on top of the server.

According to [21] a hypervisor is classified into two different types, see Figure 1. A Type 1 hypervisor is called "Bare-metal hypervisor" and it is installed directly on hardware. A Type 2 hypervisor is called "Hosted hypervisor" and runs on top of a host's operating system.

3.1.1. KVM

A Kernel-based Virtualization machine (KVM) is a virtualization software technology for the Linux kernel that makes it into a hypervisor. A KVM runs as a Linux kernel-based VM manager on Linux OS. Using KVM, an infrastructure can make multiple virtual machines running unmodified operating system images. These VMs will be managed within the host machine. KVM is an open-source software that allows running guest operating systems inside Linux processes [22]. KVM loads a kernel module into a Linux kernel as a hypervisor without creating any necessary processes like scheduler, memory manager and device drivers. These VMs running on top of the hypervisor are managed by tools to achieve live migration and resizing of the units. The architecture of a VM is shown in Figure 1. Further, KVM can virtualize the processor, RAM, network interface card, etc.

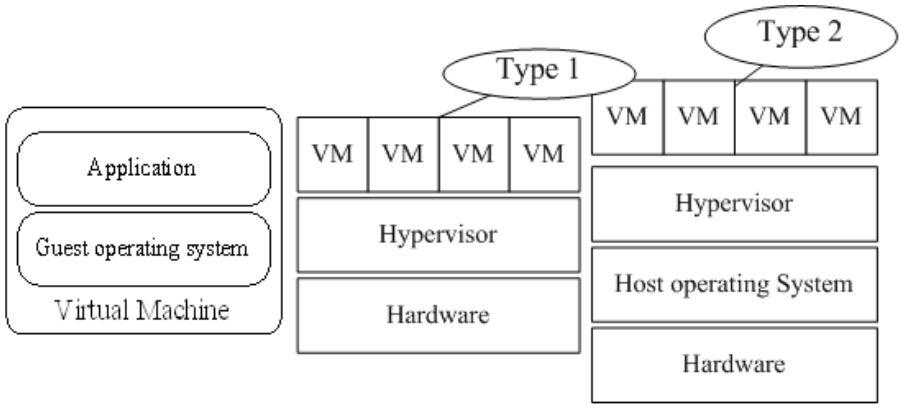

Figure 1. Virtualization Architecture: Virtual Machine and hypervisor-based architectures.

3.1.2. File system journaling

Virtual machines, expected to provide isolation, compete with the negligible overhead of lightweight virtualization using Linux containers (LXC). However, in some environments, VMs outperform LXC sharply without considering the isolation benefits. The database management system (DBMS), commonly used in clouds, provides the user with access and maintenance functions and uses virtualization for efficient resource utilization and isolation of collocated users' workloads. The disk I/O is the primary consideration when the DBMS uses virtualization technologies.

An article [26] discussed performance and isolation issues based on the virtualization technologies for DBMS, and showed that KVM outperforms LXC by up to 86% in MySQL throughput without considering the performance isolation. Due to the special conditions of the container, the isolation is one of the significant disadvantages, since the container shares the buffer caches and other data structure at the OS level. Meanwhile, journaling activities are serialised and bundled within containers, resulting in inferior performance and isolation. This performance and isolation anomaly was investigated when regarding disk I/O performance for DBMS.

The architecture and the I/O path of the journal module for the KVM and LXC are listed below in Figure 2. In KVM, the disk drives and hardware drives are virtualized with QEMU emulator. The disk I/O

request is processed inside the VM. In LXC, individual processes are running on the OS without any additional virtualization.

Therefore, the disk I/O process is handled by the ordinary process on the host. A container architecture leads to performance degradation caused by sharing of the buffer caches, and the activity inside each container will affect the performance of other containers through the shared data structure.

Multi-containers respectively update the journaling module into the transaction and commits the transaction to disk periodically or while a synchronization function is invoked, and this has a negative impact on performance isolation. The containers will be suspended to communicate the transaction and file sync from other containers, even if only one of the containers request the information update.

The file system journaling, which is regarded as the guarantee of the consistency of file systems causes the performance and isolation problem according to the measurement data.

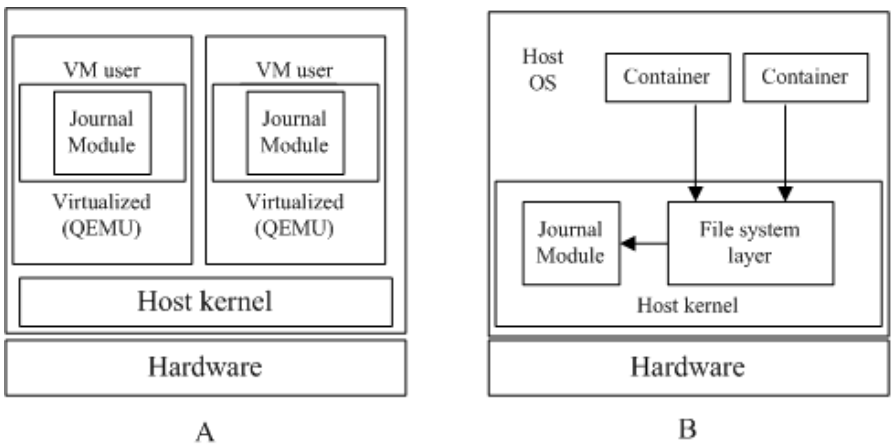

Figure 2. Virtualization Architecture: Disk I/O paths of hypervisor and container in DBMS.

3.1.3. Performance on ARM architecture

In recent days, network function virtualization, which provides a practical approach using available computer resource to improve the QoS, aims to reduce the cost and enlarge the scalability when deployed in the network by using software implementation instead of hardware deployment. Meanwhile, ARMv8 servers play an essential role due to the power consumption compare to other servers.

There is a difference when KVM is deployed on an ARM platform compared with X86 platform. The architecture for the ARM system exists on two levels, privileged exception level, which serves as the management, and a hypervisor exception level, which works for the isolation. Therefore, a KVM will be split into two levels when deployed on an ARM.

The performance when comparing VMs with lightweight deployment on an ARM architecture shows that KVM performs better on the network I/O test, which may probably due to the efficient design on ARM. For memory and CPU testing, results are quite like X86 platform where KVM has negligible overhead compared with lightweight deployment [27].

3.1.4. Xen

Xen (architecture in Figure 3) virtualization technology aims to run multiple operating systems on a signal server, and it is possible to migrate the running instance from one server to another. The user from the guest domain can choose the preferred operating system. The hardware resources will be allocated by Xen dynamically, and for security purpose, the guest OSs will not have full authority to access the computer resource. That guest operating system, which runs individually, is non-intrusive.

The guest domain (Dem U) in XEN is a virtualized environment and has few privileges interacting with other VMs. Also, there is a domain called Demain0 (Dem0), which contains the driver for all devices in the system, that has the privileges to access the hardware and handles all access to the system's I/O functions and interacts with the VMs.

The system operating overhead of XEN is much similar to KVM since they are both hypervisor-based, but in some specified environment, Xen is more complicated to deploy.

In ARM-based NFV and cloud computing, containers are an alternative with their fast deployment and lightweight execution. Even so, it still has security weaknesses compared with XEN and KVM. If there is not much performance overhead between hypervisors and containers, the isolation serves as the primary role to consider when selecting the suitable virtualization.

For this reason, Moritz Raho investigated the performance for ARM-based NFV and Cloud computing [25]. Xen is more complicated to deploy on ARM SOC, due to the changes in code and architecture. The newly added part of the code contains para-virtualised drivers and ARM virtualization extensions to achieve the function, which was initially invoked by the network emulator QEMU. Due to the changes in the architecture, most of QEMU stack was removed, making the full virtualization impossible. Para-virtualization with support by Linux and FreeBSD distribution split with frontends and backends was used to activate the virtualization in own kernel [26]. ARM hardware virtualization extensions add the HYP mode to separate the kernel from the hypervisor. The hypervisor is standalone, inside the HYP mode, taking full advantage of extensions without additional overhead. When having the performance comparison, both Xen Dom0 and DomU was considered.

The benchmark results did not show much performance overhead for the hypervisor and container, which meant that the isolation has the priority to be considered. The containers still need to consider both security and steady-state performance when deployed on a similar systematic framework.

3.2 Container-based virtualization

Compared with the hypervisor; containers have several advantages. Due to the lightweight deployment, containers may have benefits in performance overhead. Also, the system is more scalable. The VMs have higher performance costs, since multiple operating systems is running on the same host machine. However, the principle of container-based technology allows lots of guests under the unified use of a particular operating system. The system structure of the container-based system will decrease the resource consumed, thus increasing the scalability of the system.

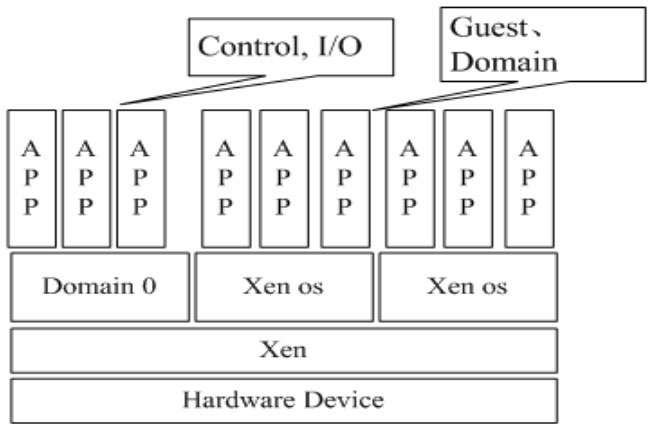

Figure 3. Virtualization architecture: Xen.

3.2.1 Linux containers (LXC)

The container is an operating system-based virtualization technology. Instead of running a complete operating system as in VMs, container access the same operating system, which includes system root files, libraries and common files. Therefore, multiple isolated processes can run in the same host. The architecture of a container is shown in Figure 4.

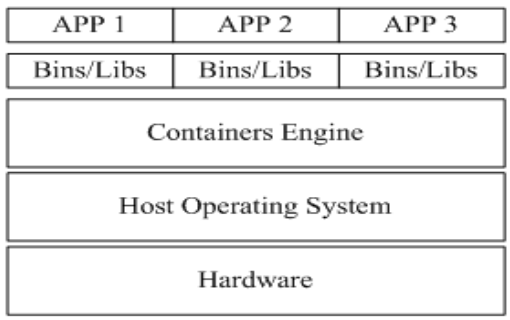

Figure 4. Virtualization Architecture: Container-based virtualization.

Linux containers (LXC) were one of the first widely used container technologies. With benefits from use space management characteristics, a user can create and manage the system or the containers more efficiently. The primary function of LXC is using a single Linux kernel virtualised on the operating system to provide multiple isolated containers.

The LXC, other than Docker, can be able to run several operating systems inside a container, managed by the namespace and Cgroups, rationally utilising the system resource. The Cgroups mainly manages the resources, including the CPU usage, memory and disk I/O limitation. On the other side, namespace takes responsibility for the resource isolation of an application's demand for the operating system. LXC shares the kernel with the operating system so that system files and running applications can be managed from the OS. Besides, the applications can sit in an isolated environment based on the property from the namespace. The architecture of the namespace and Cgroup is shown in Figure 5.

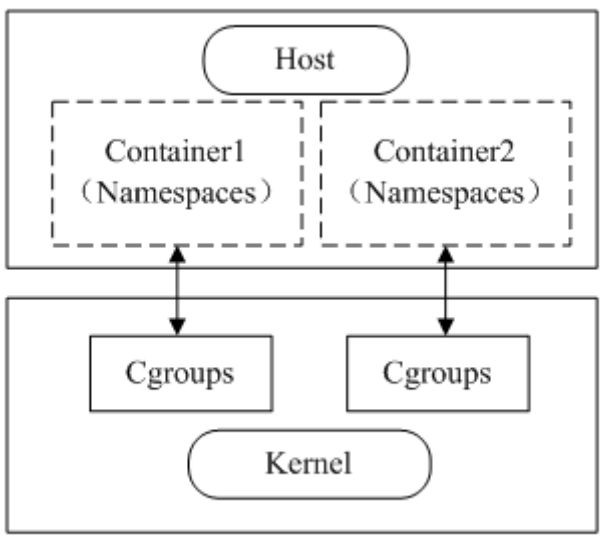

Figure 5. Virtualization Architecture: Namespace and Cgroups.

3.2.2 Containers executed on top of VM

Other than the traditional way as hypervisors and containers are deployed on cloud service, there is also a case where a container is executed on top of a virtualization machine, see Figure 6.

To compare the overhead, Ilias Mavridis [9] conducted several studies on how a container performance is affected by executed as an additional layer on top of a virtual machine. The container, described as a tool for packaging, delivering and orchestrating software service with application, has a low performance overhead when deployed. The VMs on the other hand, are doing fine in isolation. However, according to some statistical data, public cloud providers mostly offer VMs, and it is, therefore, essential to consider the lack of suitable infrastructure of a private cloud. Executing containers on top of VMs become a common case that also fulfils the enhanced security purpose. Moreover, it is easier to manage and update the system by using this technology. In the environment where a container is running on top of a VM, there is a performance dissipation, and it is recommended due to high security and easy management to execute a container on top of a VM. The extra consumption sacrificed for the security property of the system bring us an additional solution when combining both technologies for better use.

Data centres and cloud computing relies on virtualization for the public and private user. The hardware virtualization with sharing the same kernel generate lots of guest OS, which increase the consumption of the system. Replacing VMs with containers will lower the file creation of binaries and libraries, which will improve the utilisation rate.

In this case, container executing on top of VMs is not only solving the isolation and security problems but also enhancing the system. For data storage convenience, the workload migration context should be taken into consideration. It is relating to applications running as a container on a VM, which needs to migrate to other physical hosts. Two solutions mentioned in the article [25] are discussed separated. One solution is to kill the container and recreate it with the same image in the suitable host. Another solution is to replace the host VM with simulated and calculate using the mathematical mode. Depending on the theory, it is also expected to have proper timing whether to fork out the parent VM or shift new containers on some other VM. The

conclusions summarized this for a stateful container. VM migration performs better compared with the container reset and rebuild which acts on a high value of time. For stateless containers, container kill/restart performs relatively better compared to VM migration.

3.2.3 Docker

Docker container technology was created in 2013 as an open source engine. It is a tool for container management, and it was built upon LXC to package the application into the container in an efficient way. The main difference between the LXC and Docker is concering the management object, since LXC manage to run several operating systems inside a container and Docker on the other side manage signal application containers. Docker Hub as an open source platform plays an essential role in supporting container images to the applications and services. [23].

Compared with VMs, containers have much less overhead because of the sharing system that decreases the image size and reduce resource consumption of the infrastructure. The architecture is shown in Figure 7. The basic principle of Docker is to pull the images from the libraries and pack them into the containers. Docker consists with three parts: Docker Host, Docker Client and Registry.

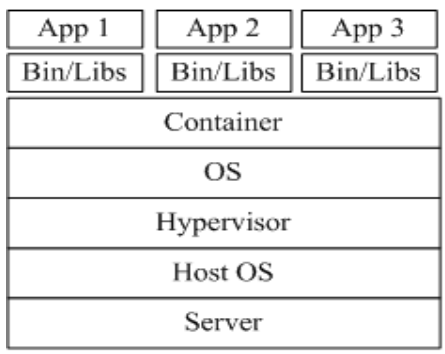

Figure 6. Virtualization Architecture: Container on top VM.

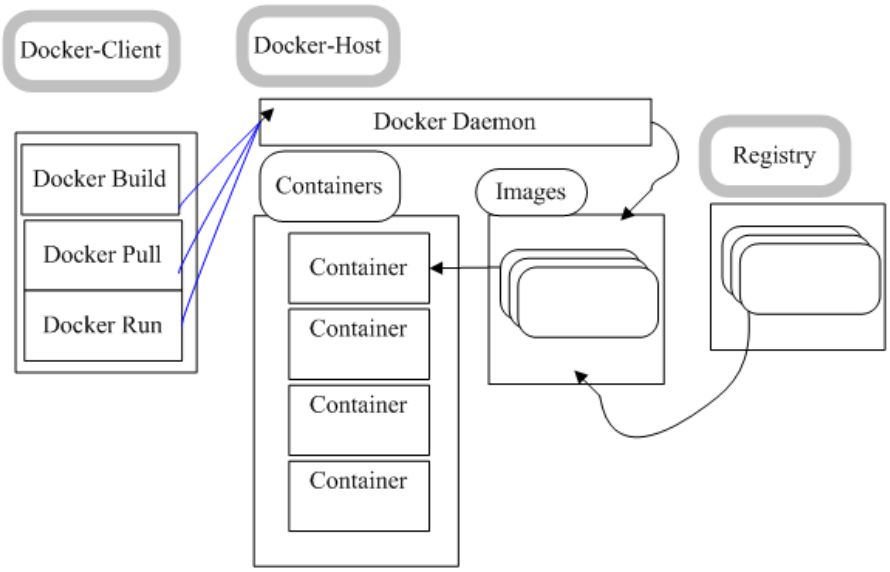

Figure 7. Virtualization Architecture: Docker

With the copy-on-write (COW) file system, Docker instantiates containers faster. The pointer directly invokes the existing files and package them into the container. Also, the file system supports the layering structure, where the container within the structure can operate based on the containers contain with the existing requisite files.

The COW file system has a unique process when handling the live data. Instead of overwriting the data, the system updates the live data using the unused blocks on the disk. The access authorisation is postponed until all the data is updated to the disk. Copy-on-write ensures the repetitive resource usage by multiple tasks and saves memory and CPU resources. When several applications request for the same data, only one memory space is allowed by the COW, and this memory space points to all the apps [8]. If the application requests to modify the data, the new data will be uploaded to unused block in the disk attached with its independent memory space. The rest of the applications continue to call the data functions with the original pointers.

The isolation of Docker depends on the layer between the applications. Furthermore, the layer between the application and the host restricts the access to the host. The containers can only access the directory structure so that each container run individually. Even though each container operates independently, the application inside still has full of its dependencies like the libraries and share the same kernel with other containers. Finally, containers run the isolated process in the user-space on the host operating system.

3.2.4 Memory resource management

When container-based virtualization became popular in recent years, many web developers and web application companies tend to use it as a primary platform for their applications. Taking good advantage of hosting multi-applications, leads to considering resource consume problem such as memory consumption.

Unexpected access concentration in a target system causes additional memory consumption, resulting in shortage on the host. Besides the application bug related to the incorrect memory allocation leads to an unexpected memory leak. A paper [4] investigating the problem proposes that memory overuse is prevented by estimating the memory usage of each container.

The memory management method for container-based virtualization solves the memory consumption by setting the limitation for the memory usage of the containers, since the problem a container may have is a memory overuse feature. Without a memory monitor, operating either limits the memory resource for each container to prevent overuse even when some of the containers are out of control or terminate the random process until the host system restores to regular ownership of the memory to decrease the degradation of the performance. Decreasing the number of containers managed by the host if neither of the schemes works is another solution. Meanwhile, the execution time handling the memory problem may still affect the availability of the host.

Setting the consumption of each container as an index to evaluate the behaviour, will monitor the memory usage and limit the consumption when an unexpected threshold value pops up. Certainly, the system detection will periodically examine the resource-restricted

containers to release the memory until the container memory consumption becomes normal again.

3.2.5 Container performance comparison in NFV

The operator will consider the selectivity for virtualization in the different environment based on the performance and complexity rate for deployment and maintenance.

Containers, which are widely used in NFV based applications, will cause significantly improvement, especially in large-scale microservice architectures such as telecommunication companies when combined with the best performing containers and network. The function for NFVs is mainly for application handling the package in the network, primarily impacting on the parameters of the average packet per second and store-forward latency.

In container-based technology, there are several choices to be considered. Jakob Struye [23] contributes with a detailed comparison of three container-based technology providers' network performance impacting the network function virtualization. The traffic throughput and network latency were affected by different container technologies deployed in the NFV based network. Three main container implementations, including LXC, DOCKER and RKT were deployed individually.

Running one or more containerized user datagram protocol forwarding NFVs on host machines concluded that RKT had the worst performance and LXC had the best. For packet processing, spread over different applications with a different network solution, there is a latency improvement compared with the bare metal machine. In additional, if we only consider the network solutions that are processing the NFVs, VLAN networking has 20% gain compared with host networking, where separate buffer for each processor may gain the advantage.

LXC can provide the significant performance gain even if the maintaining and deploying is more complicated than for Docker and RKT. It is not hard to work out that Docker has advantages implementing on NFV comparing with the others using statistical analysis.

3.3 Network Function Virtualization and Software Defined Network

For the future 5G network, one of the essential technologies is to utilise the network function virtualization. The NFV virtualizes the network components to achieve flexibility and lost cost solutions for network services.

To overcome the challenges of the huge increasing numbers of network traffic, building more infrastructures is one of the hardware-based solutions. However, this is restricted due to scalability issues and huge costs. One of the alternative solutions called Software-Defined Network (SDN) or Network Function Virtualization (NFV) is moving forward. To save the resources and avoiding machines working for the limited purpose, NFV enables decoupling the required network functions from the specific hardware devices, and is a software-defined virtualizing to handle the traffic on demand. Meanwhile, NFV has more parameters to consider when deployed in the network. Compared with purpose-built machines, NFV is a software-based solution to maximize performance from existing devices. It is essential to calculate the processing capacity of the NFV and have a good balance with the workload. Furthermore, NFV is managed dynamically. Therefore, it may cause network congestion due to delays or randomly high throughput.

In real-world applications, the important technologies that virtualized the network function are related to hypervisor and container, both virtualized technologies have advantages in a certain field.

3.3.1 Virtualization Comparison under Open5GCore

The Open5GCore is a commercial product for software-defined network evolved packet core (EPC) over hypervisor and container.

The advantage of NFV is that it enables executing several virtual machines at the same time, and this is controlled by the system. Furthermore, according to the state of the network, an NFV instance will adjust the computing and capacities in real time. Except for NFV, it is also quite interesting to investigate virtualized technologies that have a good performance on carrier-grade networks.

During the installation of Open5Gcore, it is essential to set system privilege mode for Docker to change the configuration of the network

interfaces, because Docker is deployed on application level and cannot directly access libraries and host OS. However, it is hindering the process from setting the system permissions. The privileged mode situation leads to the security issue.

A research article about the evaluation of Open5GCore over hypervisor and container [16] compared the CPU and memory usage to show the resource utilisation during the evaluation stage. Meanwhile, VoIP, Video, and file transfer protocol (FTP) profiling was used to show network performance.

Calculation examples show that KVM has the highest CPU utilisation, which is due to executing both host OS and guest OS. For memory usage, KVM has the highest performance, and the memory will not increase too much as the background traffic is growing as the required memory is allocated when a virtual machine is created. The network performance comprises the throughput and delay. In the testing stage, the physical machine has the lowest delay and KVM has the largest delay.

Video, VoIP and FTP stand for small and large data transmission respectively. In light traffic, physical machine KVM and Docker have almost the same throughput. This is probably caused by excessive CPU and memory assembled on the local machine. For the FTP scenario; the physical machine is the best due to the lowest overhead among them.

The throughput between the physical machine and KVM/Docker are close, and benefit by using the same interface virtualization technique, macvtap. The gap throughput between the physical machines and KVM/Docker can be as large as 3Mps. Finally, regarding the availability of the system, the boot time, reboot time, and recovery time of Docker are much shorter than for KVM.

3.3.2 Network traffic simulation with Open5GMTC

The open5GMTC is a protocol type tool for LOT/M2M platforms. The main function of open5GMTC is a generator for simulating the different kinds of network traffic to evaluate the performance between the physical machine and virtualized machine (KVM, Docker)'s Evolved Packet Core (EPC). The openMTC platform consists of two service capability layers: a gateway service capability layer (GSCL) and a network service capability layer (NSCL). The open5GMTC is an

application running on top of the openMTC. LOT/M2M was introduced as a new paradigm for real-world applications connecting to the Internet to improve the quality of the service

It is necessary to set the system mode to privileged mode and then set the network mode to none when installing the open5Gcore on a container, since the container is an application-level process and configuration cannot be settled when the system is running.

Four application signals simulated by the generator (Open5GMTC) were created after the installation. The eHealth traffic represents small bandwidth but high reliability and extremely low latency. The video traffic requires high bandwidth, but long delay and low reliability is tolerated. Luggage traffic simulates a Bluetooth low energy tag for tracking purposes and it needs high reliability but can tolerate a long latency. The last signal is called smart meter data, which contain location and device ID, timestamp and meter reading, and it requires high availability but low bandwidth. The generator can set the different payload size and frequency of requests for the different kinds of traffic pattern. By processing, the data plane traffic evaluates the performance of different virtualized EPCs.

When investigating the performance evaluation of open5GCore over KVM and Docker by using open5GMTC [15], it is concluded that the virtualization of EPC, especially Docker, is feasible because of the low overhead. The virtualization needs more memory to maintain the VMs and containers, and KVM has more massive memory usage than Docker, since Docker is only virtualized on the application level. Even though KVM and Docker consume more CPU resource， the CPU consumption between the virtualized system and physical machine has not much difference. Virtualized EPC on KVM and Docker can have almost the same performance as the physical machine if the number of the devices they serve is less than 70.

3.4 Benchmark Application

This section describes benchmark tools that are used to measure the performance overhead of hypervisor and container compared with non-virtualized bare metal. In general, parameters used to process the data comparison are listed below. The testing data used to analyse the performance are computing power, data transfer rate, memory usage and data flow, which represent CPU, disk, memory and network respectively. Both virtualisation technologies were tested independently compared with the non-virtualized environment. More remarkable, there is much difference between the benchmark tools impacting on the system performance compared with the CPU performance.

3.4.1 System performance

Several benchmark suits are available for testing the overall performance of a system. The various aspects of the system result in the raw score gives the most intuitionistic exhibition. The index value stands for the interaction of different parts of the system.

3.4.2 CPU performance

The method of calculating the CPU performance is to run extensive tests that estimate the prime numbers up to the limitation of the system, and it is default to run a signal thread for execution on top of every solution. Several measurement results are used to calculate the average value and standard deviation in order to show the performance of the host machine under the different virtualised environment.

1) LINPACK: The principle of LINPACK [9] is to measure the computer's floating-point rate of execution and this is determined by a program that solves a dense system of linear equations. As a collection of Fortran subroutines, LINPACK solves the equation with the function "Ax = b" (random matrix A size N, vector b) by performing lower-upper decomposition of numerical analysis with partial pivoting.

2) Y-cruncher: Y-cruncher [17] is a multi-threaded benchmark tool for a multi-core system to calculate the value of Pi. As a constant number computed or generated by the system,

Pi is used as the stress testing parameters for the application estimating the performance of CPU. This benchmark tool is resulting in several outputs including multi-core efficiency, computation time and total time.

3.4.3 Memory performance

1) STREAM: The STREAM benchmark is a simple synthetic program that measures the sustainable memory bandwidth in Mb/s and the computing rate, which is related to the vector kernels. While we perform calculating, four simple vector operations, Copy, Scale, Add, and Triad, perform as the primary operating parameters. These four operations are listed in Figure 8. As an addition, the performance is measured has a strong dependency to CPU cache size; it is recommended to set the "STREAM array" size properly. The arrays are required to be much larger than the most significant cache(s) used to ensure the data validity.

3.4.3 Disk I/O performance

Disk performance parameters are processed by changing the file size simulating the file handing situation in the different environments based on different virtualisation. The file system benchmark is used to help analysis and calculating. In the file system operations, read/re-read, write/re-write, and random read /write, are the main comparison subjects. Some articles showed that hypervisors have better performance in disk I/O due to the cache mechanisms.

1) IOzone： IOzone is an open source solution, and it is a file system benchmark tool that is extensively used to perform analysis in almost all main platforms such as Linux, BSD, MacOSX and Windows. IOzone was developed by William Norcott and then enhanced by Don Capps. Basically, IOzone runs the testing on a default file and generates the data based on reading, writing and random read/write.

2) Bonnie++： Bonnie++ is a benchmark tool for Unix-like operating system, developed by Russell Coker. It can handle the testing file for more than 2G on a 32bit machine, and the operations include create (), stat (), and unlink (). There are two significant effectiveness testing details including system file I/O test and file creation tests. In the File I/O test, sequential output, sequential input and random seeks are processed respectively.

Operations	Kernel
Copy	x[i] = y[i]
Scale	x[i] = q * y[i]
Add	x[i] = y[i] + z[i]
Triad	x[i] = y[i] + q * z[i]

Figure 8. Benchmark STREAM operations.

3.4.4 Network throughput

For testing purpose, one of the effective ways to measure network throughput is setting the traffic generator by simulating sending and receiving data from emulated devices, calculating the target parameter. The traffic generator also allows setting the appropriate payload size and frequency for different traffic pattern with the modified value of payload size, generator frequency and the undefined emulated device simulating the testing scenarios.

1) Netperf: As a benchmark tool, Netperf can be used to measure various types of networking. It can measure the unidirectional throughput and end-to-end latency for giving purpose. By using the BSD sockets, Netperf performs the measurements for TCP and UDP for both IPv4 and IPv6. Also, Netperf has compatibility with Unix Domain Sockets [19].

2) Iperf : Iperf, which quality is quite close to the industry standard is used to measure the maximum utilisable bandwidth between a server and a client. It supports various parameters including timing, buffers and protocols (TCP, UDP, SCTP with IPv4 and IPv6), and reports the bandwidth, package loss and other parameters.

4. Experimental part

4.1 Testbed

The main objective of the experiments is to investigate the performance difference when each virtualization approach is implemented in different working environments. The gains or losses of hypervisor-based and container-based virtualization when implemented in a real-world application should be analysed and calculated in an experimental context. Our target is to explore the performance distinction when a web server is deployed on a hypervisor-based or container-based virtualization environment. This should be compared with system resource consumptions in different virtualized environment to provide the theoretical basis of analysis. For the performance comparison, two different types of virtualization, KVM and Docker, were selected to represent the hypervisor-based and container-based virtualization technologies. Both virtualization systems are executed on the Ubuntu 18.4 system to ensure the same testing environment. Also, they utilize the same hardware resources from the host machine, with an 8 core, 2.3 GHz CPU, and 16GB memory.

The testbed consists of a host machine and two virtual machines. When the interaction happens, it is available to monitor the functioning capability of each virtual machine on the host machine, and the load environment can be modified on the host machine to achieve the convenient data adjustment. The testbed is designed to simulate the load capacity of the webserver when running on a different virtualized environment. The load test tool is deployed on the host machine, and the target server is implemented on the guest machine.

In the testing scenario, the software imitates the customer and send the HTTP request to the webserver. The webserver responses and deal with the Http request with the system resources consumed.

Our primary objective is to determine the system performance while handling a large number of the HTTP requests in a short period. Since the webserver is executed on different virtualization environments, there will be different performances depending on the different virtualizing framework.

A similar work to ours is the virtualized MPTCP proxy performance comparison in [10]. Their studies were mainly focused on the virtualization proxy that implements the multipath TCP connection to retransmit the data to the TCP-based host. The proxy server respectively deploys the KVM and Docker to measure the elapsed time and data throughput. With a similar concept, we build the testbed and mainly focus on the system resource consume. A webserver deployed in different virtualization environments, will result in a performance difference when doing the load test. The host machine act as a transmitter and the guest machine receives the data package.

4.1.1 Host machines deployment

The testbed mainly consists of the host machine and the virtual machine, which respectively represents the clients and the webserver. Besides, several benchmark tools on the host machine are used to monitor the overall information of the system. The host machine represents the transmitting side and simulates multiple clients. The clients will access the webserver in a short period for testing purpose. The host machine can achieve the Http emulation by sending numerous requests to the webserver, which is deployed on the virtual machine. Also, the benchmark tool will monitor the system performance while the Http emulation is running to obtain the real-time data of the system.

The Apache JMeter deployed on the host machine is used to load functional test behaviour. Modifying the transmitting frequency and time will affect the load capacity. The Apache JMeter is running on the host machine to keep sending the requests to the terminal server, emulating that the client establishes the data connection while the network is busy. When increasing the request number that is sent from the transmitting side, the performance changed due to the load testing.

The load capacity of the virtual server is visualized by the "HTOP" monitor, which visualizes the utilization of the system resources separately. The webserver is running on top of the virtual machine and can be reached by the pre-allocated static IP address given by the router. By using the network configuring rules of the bridge-connection when creating the virtual machine, enables the host machine to directly connect and access the guest machine. Otherwise, the guest machine, as an independent device, can only establish the data connection with other equipment except for the host.

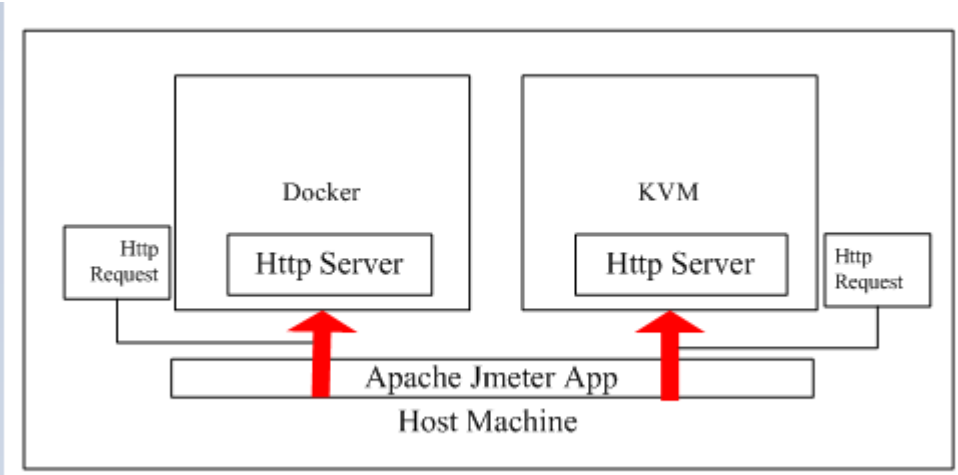

Figure 9. Testbed architecture for load test.

Two virtual machines are running on top of the host machine. The webserver will set up inside of the virtual machine. For comparison purpose, two different virtualizations KVM and Docker respectively represent the Hypervisor and Container virtualization. The simulation system is in Figure 9.

The test objects are running on top of the host machine with different virtualization methods. Both virtual machines are running the Ubuntu 18.04 system to ensure the same testing environment. Moreover, both virtual machines allocate the same system resources. To achieve the function of a webserver, we install the Apache HTTP server on both virtual machines. The experiment happens between the host and guest machine and emulates the network scenario. The firewall (UFW) rules need to be modified so that they ensure the successful establishment between the host and the guest machine. The transmitter sends the HTTP request to the receiver, and the web server deployed on the virtual machine responses to the application following the network rules as what happened in the real world.

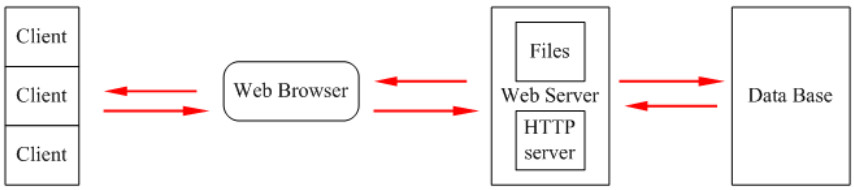

Figure 10. Architectural of the webserver principles.

The virtual machines created with the hypervisor and container virtualization methods affect the overall performance of the webserver while the available system resource maintains the same. With the pre-allocated static IP address given by the router, the host machine can connect and access to the guest machine. The simulation scenario assumes that the client, or the customers randomly access to the webserver, and that the server processes the network traffic based on the hardware resources. The experiment data will show the webserver performance when running under different network traffic situations and availability of system resources.

All the benchmark tools and system versions of the experiment testbed are listed in Figure 11. The Docker image version of the Ubuntu system keeps the same as the KVM and the host machine.

property	Tool	Version
Container	Ubuntu	18.04
Hypervisor	Ubuntu	18.04
Http generator	Apache Jmeter	5.11
Web server deployment	Apache2	2.4
Cpu usage monitor	Htop	2.20

Figure 11. System inventory of the testing environment.

4.2 Experiments

The experiments are including the host and the guest machine. With the help of Apache JMeter, we can emulate the network scenario in a different data flow. The primary research questions include the system resource consumption when the webserver is deployed on different virtualization environments and the reacting time when the webserver is handling massive amounts of Http requests in a short time period. These questions reflect the virtualization performance when the real-world software is running on top of a hypervisor and a virtual machine, and utilize the system resources based on the different characteristic of the virtualization method. Multiple tests will be executed in order to ensure the reliability of the experimental data.

The simulation starts with the Http generator, and the thread group plays a vital role in the experiment stage. The number of threads and the ramp-up period settled in the test plan and we observe the overall system performance in order to analyse the variety in the different virtualization frames.

We set the thread numbers to correspond to the network situation in a certain period. Moreover, the thread numbers represent, respectively, light traffic, medium traffic and heavy traffic, see Figure 12. The result is shown in form of graphics, and our objective is to observe the computer resource utilization. We neglected the response time and the error rate, which are the main parameters to test the quality of a website. One reason is that the simulation is running inside the computer and that will greatly reduce the response time and packet loss rate. We only focus on the CPU consumption when the server handles the HTTP request in the different virtualization environments.

Here we mainly focus on the performance variation under all kinds of system resource availability. The load test increases the occupation of the system resources and the web servers under different virtual environment will have different expressiveness.

Traffic situation	Thread Number/1s
Light	1000
Medium	5000-7000
Heavy	10000

Figure 12. The experiment parameter of the thread group.

4.2.1 System's resource consumption

The operation of the webserver will result in an overall system performance that concerns one of the research questions. The question tries to do a tentative discussion about the overall system performance when the webserver is under different virtualization environments. We recorded the CPU usage of the 'Htop' benchmark tool when the system is emulating the Http interaction, and the webserver is handling the network traffic. Here we focus on the total usage of CPU core because the consumption by the HTTP emulating software is always the same no matter which virtualized server is used. The gap of the numerical value represents the performance difference between the two virtualization methods.

4.2.2 Elapsed time of the response

Another parameter used to analyse the webserver state is the average reacting time when the interaction happens between the host machine and the webserver. The response time represents how fast the server is reacting and handling the data traffic. The average connecting time, and the deviation will be used when discussing the KVM and Docker individually.

In this experimental stage, the data will indicate the response time when the webserver, which runs on top of KVM or Docker, handles the client's application. We mainly focus on the reacting speed when the system resource is limited and hence the sampling number is set to 10 000.

5. Results

In this chapter, the results from the experiments are presented and discussed.

5.1 Light Traffic

The first experiments show that when there is light traffic, which means that the task of the process is not substantial enough; the performance of two virtualization methods are very similar. We set the thread number on the HTTP generator to 1000 during a second and the virtual machine can handle the requests easily. The monitor interface on the ‘Htop’ shows precisely the amount of CPU usage while the interaction happens between the 'host machine' and the 'guest'.

According to the record, the KVM virtual machine has almost the same consumption comparing with the Docker container when the system task is not in an oppressive situation. Both virtualization methods have enough computing resources. The usage of the CPU is around 15.9% for both multi-thread tests under different virtualization environments.

5.2 Medium Traffic

According to the test plan, the thread number needs to be increased in order to determine the web server state when the virtual system is under network stress. We set the threads number as follows: medium (5000-7000 threads) and heavy (10000 threads) traffic during one second for performance comparison purposes.

The results for medium traffic are shown in Figure 13. Even if the Http requests access to the website during 1 second, for some reasons, the system still takes around 2 seconds to complete all the requests. The CPU consumption rapidly increases to a peak value and then drops to normal. Here, we are running experiments 5 to 10 times in a row and record the average value of the CPU consumption.

The results show the CPU consumption under medium network traffic load. The benchmark tool records the peak, and total CPU usage when the HTTP generator keeps sending medium traffic to the

webserver. As the number of threads increase, the CPU becomes under pressure, and there is a performance gap between the KVM and Docker. The results show that Docker has around 12% less consumption compared with KVM. The system takes advantage of the lightweight execution of the virtualized system architecture, and, therefore, Docker has a better performance when the system resources are limited.

5.3 Heavy traffic

For Heavy traffic (10.000 threads), the results may be affected of the computer capacity. As we increase the thread number to 10.000, the system is under heavy pressure, and Docker has not much advantage compared with KVM in this kind of testing environment. In addition, the Http generator utilizes tremendous system resources as well. Therefore, the testing result is probably not accurate, since the experiment shows that there are insufficient system resources. However, according to the result, shown in Figure 14, Docker still has some advantages.

As can be seen in Figure 14, Docker has a 3% advantage during the heavy network traffic load. While the CPU is under immense pressure, the result has an inaccuracy. The test should have been executed in an environment where the system resource is enough to ensure the accuracy of the result.

Total Cpu usage		
Threads	KVM	Docker
5000	45.8%	29%
7000	77.75%	64.35%

Figure 13. CPU consumption under medium traffic load.

Total Cpu usage		
Threads	KVM	Docker
10000	83.1%	80.3%

Figure 14. CPU consumption under heavy traffic load.

5.4 Average response time

Another index we measured within the experiment was the response or the reacting time of the web browser. Unlike the overall system performance comparison, this index shows how fast the virtualization responses to the HTTP request when the system resources are enough. The average response time is in millisecond scale, and the deviation indicates the stability when the connection is established between the host machine and the webserver. The average response time and the deviation are listed in Figure 15. This experiment was executed with 10.000 connections to the webserver.

The measurement happened inside the host machine, so the response time purely indicates the reacting speed when the webserver is running in different virtualization environments without the interference from outside. In Figure 15, we can see that Docker is more stable when responding to the connecting requests, since the average response time and the deviation is less than for the KVM virtualization machine. These results should be combined with the results in the stress testing, where sampling 10.000 HTTP requests consumed around 80% of the system resources. The Docker reacts to the responses faster and more stable when there is an insufficient amount of computing resources.

The lightweight virtualization method, like Docker, will have an advantage when comparing the total data throughput, no matter what traffic load that arrives at the system.

Virtualization environment	Average response time/ms	Deviation
KVM	7	8
Docker	2	2

Figure 15. Responses time and deviation of the webserver.

6. Conclusions

In this thesis, we evaluate the performance impacts of representative hypervisor-based virtualization, KVM, and container-based virtualization, Docker. The thesis investigates the system performance when a webserver is deployed on top of one of the virtualization methods. We implement testbed with an HTTP transmission emulating the real-world implementation of different virtualization frameworks and evaluate the system consumption and server response time, respectively. The benchmark tools are computing the overall system performance and network state when the experiment happens. By analysing the experiment results, we can conclude the following things.

Executing a webserver on top of a virtualization environment is common these days, and the data traffic is affected by a variety of factors, but not limited to the virtualization method. Many kinds of researches have compared the performance between a hypervisor and container respectively and how they impact on individual components. Our goal is to contrast the theoretical expenditures of the operating system with an actual situation where the application is working in different virtualization environments. For a target customer, cost factors should be taken into consideration when designing the virtualization system. As none of the virtualizations are beneficial absolutely, the operator should consider the operating situation and make the right choice to lower the overall costs and improve the quality of service.

In theoretical studies, some articles support that KVM has a robust system isolation architecture. A container running with root privileges makes the system unreliable. The isolation becomes one of the main advantages when comparing KVM with a Docker virtualization environment. The isolation property makes it feasible of running Docker on top of VMs, which can achieve both system isolation and performance improvement [9]. Besides, KVM performs better when the memory needs more frequent access, and when handling the intermediate data that stored in memory instead of the disk [12]. The situation has decreased memory usage as the Docker share all system resources instead of KVM that have distributed the memory resource to the guest operating system in advance.

Following the experiment relating to the KVM and Docker, we conclude mainly on the performance of the virtualization concerning the real-world application. The theory part points out how virtualization is affecting the individual components of the hardware resources. The performance comparison is performed when the system is executing under different virtualization environments. The CPU consumption, memory usage or network throughput are separately discussed with a specific benchmark method. In our test stage, we focus on the overall performance of the virtualization environment. The real-world application is running in a different background and have interdependences based on the virtual system.

In our experiment part, we mainly focus on the webserver state when executing in different virtual environments. Here we conclude that the container's average performance is generally better than the hypervisor virtualization. The operational steps when deploying a webserver on a virtual machine is more complicated than when the Docker environment is used. The network settings, which are default as the NAT on KVM needs to specify the bridge mode. On the other hand, Docker simplifies the steps of network configuration. Two questions are specifically discussed based on our testbed. The first question is the system overall consumption when the HTTP server is deployed on the KVM and Docker environments. The second question is the response time when the webserver is running in different virtualization environments.

In the experiment stage, we specifically discuss three kinds of network situations. KVM has a similar performance with Docker only when the system resources are enough, or the network traffic is light. In the rest of the testing stages, Docker has around 10% less overhead compared with KVM when both virtualizations occupy equally amounts of hardware resources. Another measurement concerns the response time when an HTTP server is running in different virtualization environments. The deviation represents the stability of the HTTP server. In the test environment, Docker achieves excellent results compared with KVM. Also, the Docker environment has more advantages in response times compared with hypervisor-based virtualization. When the performance is the only consideration, Docker takes advantages based on the behaviour of the network state when deploying the HTTP server on it. Since our experiments relate to the

network handling, memory usage and CPU consume, according to the test results, Docker has advantages in the relevant aspects.

As a further work, it would be interesting to investigate the combination of two virtualizations when running Docker on top of VMs. The topic relates to how additional virtualization layers affect the overall system performance, resource utilization and network state. Since there is no comprehensive solution offering this virtualization scheme, combining these two virtualizations may produce better results.

References

[1] Zheng Li, Maria Kihl, Qinghua Lu and Jens A. Andersson (2017). Performance Overhead Comparison between Hypervisor and Container based Virtualization. Department of Electrical and Information Technology, Lund University, Lund, Sweden.

[2] Roberto Morabito, Jimmy Kjällman, and Miika Komu（2015） Hypervisors vs. Lightweight Virtualization: a Performance Comparison, Ericsson Research, Nomadic Lab Jorvas, Finland.

[3] The next hypervisor LXD is fast, secure container management for Linux Cloud. [Online]. Available at: http://coreos.com/blog/rocket/, last accessed 12/Dec/2014.

[4] Gaku Nakagawa, Shuichi Oikawa "Behavior-based Memory Resource Management for Container-based Virtualization" Department of Computer Science University of Tsukuba Tsukuba, Ibaraki, JAPAN.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in SOSP '03. ACM, 2003, pp. 164–177.

[6] A. Sampathkumar（2013） "Virtualizing intelligent river R: A comparative study of alternative virtualization technologies" Master's thesis, Clemson University.

[7] Bo Wang, Ying Song, Xiao Cui, and Jie Cao （ 2017), Performance Comparison between Hypervisor- and Container-based Virtualizations for Cloud Users Software Engineering College, Zhengzhou University of Light Industry, Zhenzhou, China, 450002

[8] Prof. Ann Mary Joy (2015) Performance Comparison Between Linux Container and Virtual Machine. IMS Engineering College Ghaziabad India.

[9] Llias Mavridis, Helen Karatza (2017) Performance and overhead study of containers Running on top of Virtual Machines.

[10] Sunyoung Chung, Seonghoon Moon, Songkuk Kim（2016） The Virtualized MPTCP proxy performance in Cellular Network, Yonsei Institute of Convergence Technology Yonsei University.

[11] Wubin Li (2015) Comparing Container versus Virtual Machines for Achieving High Availability.

[12] Janki Bhimani，Zhengyu Yang, Miriam Leeser, and Ningfang Mi （2017), Accelerating Big Data Applications Using Lightweight Virtualization Framework on Enterprise Cloud Dept. of Electrical & Computer Engineering, Northeastern University, Boston, MA USA.

[13] Domenico Cotroneo, Luigi De Simone, and Roberto Natella （2017） NFV-Bench: A Dependability Benchmark for Network Function Virtualization Systems.

[14]https://www.ibm.com/developerworks/aix/library/auaixhpvirtualization/, [Online; accessed 26-December -2018].

[15] Hung-Cheng Chang, "Performance Evaluation of Open5GCore over KVM and Docker by Using Open5GMTC".

[16] Hung-Cheng Chang, "Empirical Experience and Experimental Evaluation of Open5GCore over Hypervisor and Container" 2018.

[17] http://www.numberworld.org/y-cruncher/ [Online accessed 30-January-2019].

[18] https://www.coker.com.au/bonnie++/readme.html/; [Online accessed 29-Jamuary-2019].

[19] https://hewlettpackard.github.io/netperf/ . [Online; accessed 28-Jamuary-2019].

[20] Andrey Mirkin, Alexey Kuznetsov, and Kir Kolyshkin. Containers Checkpointing and Live Migration. In Proceedings of the Linux Symposium, pages 85–92, 2008.

[21] Popek, Gerald J., and Robert P. Goldberg. "Formal requirements for virtualizable third generation architectures." Communications of the ACM 17.7 (1974): 412-421.

[22] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” in Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on. IEEE, 2015, pp. 171–172.

[23]https://www.docker.com/products/docker-hub/. [Online; accessed 26-December-2018].

[24] A. Polvi, “CoreOS is building a container runtime, rkt,” https://coreos. com/blog/rocket.html, 2014, [Online; accessed 21-December-2018].

[25] Y.C.Tay, Kumar Gaurav, Pavan Karku （ 2017 ） A Performance Comparison of Containers and Virtual Machines in Workload Migration Context National University of Singapore.

[26] Asraa Abdulrazak Ali Mardan, Kenji Kono (2016) Containers or Hypervisors, Which is Better for Database Consolidation?, Department of Information and Computer Science Keio University .