Tribhuvan University
Institute of Science and Technology

A Performance Comparison between Hypervisors and Lightweight Virtualization

Dissertation

Submitted to
Central Department of Computer Science and Information Technology
Kirtipur, Kathmandu, Nepal

In partial fulfillment of the requirements
for the Master's Degree in Computer Science and Information Technology

By
Salina Shrestha

April, 2019

Tribhuvan University
Institute of Science and Technology

A Performance Comparison between Hypervisors and Lightweight Virtualization

Dissertation

Submitted to
Central Department of Computer Science and Information Technology
Kirtipur, Kathmandu, Nepal

In partial fulfillment of the requirements
for the Master's Degree in Computer Science and Information Technology

By
Salina Shrestha

Supervisor
Prof. Dr. Subarna Shakya

April, 2019

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

Student's Declaration

I hereby declare that I am the only author of this work and that no sources other than the listed here have been used in this work.

…………………
Salina Shrestha
Date:

Supervisor's Recommendation

I hereby recommend that this dissertation prepared under my supervision by **Ms. Salina Shrestha** entitled " **A Performance Comparison between Hypervisors and Lightweight Virtualization**" in partial fulfillment of the requirements for the degree of M.Sc. in Computer Science and Information Technology be processed for the evaluation.

……………………………
Prof. Dr. Subarna Shakya
Department of Electronics and Computer Engineering
Institute of Engineering, Pulchowk Campus
Lalitpur, Nepal
Date:

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

LETTER OF APPROVAL

We certify that we have read this dissertation and in our opinion it is satisfactory in the scope and quality as a dissertation in the partial fulfillment for the requirement of Master's Degree in Computer Science and Information Technology.

Evaluation Committee

…………………………………	……………………………..
Asst. Prof. Nawaraj Paudel	**Prof. Dr. Subarna Shakya**
Central Department of Computer Science and Information Technology	Department of Electronics and Computer Engineering
Tribhuvan University, Kirtipur, Nepal	IOE, Lalitpur, Nepal
(Head of Department)	**(Supervisor)**
……………………………..	…………………………..
Mr. Lochan Lal Amatya	**Mr. Tej Bahadur Shahi**
(External Examiner)	**(Internal Examiner)**

Date:

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my supervisor **Prof. Dr. Subarna Shakya.** I would like to thank him for his guidance, patience, insightful discussions, trust and correction of my thesis work so that my dissertation was possible.

Additionally, I would like to thank **Asst. Prof. Nawaraj Paudel**, Head of Central Department of Computer Science and Information Technology, Tribhuvan University for his encouragement and guidance.

I would like to express sincere thanks to all the faculty members and staffs of Central Department of Computer Science and Information Technology, Tribhuvan University who helped me in any possible way to complete my thesis work. I am indebted to all the people who supported me and encouraged me directly or indirectly to complete this work.

Last but not the least, I would like to take this opportunity to thank my family for their love and supporting me throughout my life.

ABSTRACT

The term virtualization usually implies talking about hypervisor-based virtualization. However, in the past few years, lightweight virtualization technologies claim to offer superior performance. Virtual machines offer high flexibility and easier management. They also enable flexible scaling, which makes it easier to respond to the varying traffic patterns. But, the traditional virtual machines comes at the cost of overhead and have reduced performance in most of the operations. One high performing alternative to a virtual machine is a Linux container. Containers are isolated user spaces which share host computer's kernel. This makes processes inside them perform almost as well as if they would be running directly on host. A detailed performance comparison between traditional hypervisor based virtualization and lightweight virtualization has been presented. Comparison has been done between KVM and LXC/LXD. The base OS for both technology is Linux since LXC only allows Linux as guest operating systems. Benchmarking tools and script have been used for the measurements to understand the performance in terms of processing, memory, storage and response time of web server. The result after the comparison show that the Linux containers achieve better performance when compared with traditional virtual machines.

Keywords: Performance, Virtualization, Hypervisors, Container, Benchmarking.

dedicated to my parents…

Table of Contents

List of Figures

List of Tables

List of Abbreviations

VM	Virtual Machine
VE	Virtualization Engines
API	Application Programming Interface
KVM	Kernel Virtual Machine
LXC	Linux Containers
HTTP	Hyper Text Transfer protocol
IPC	Inter-process Communication
HPC	High Performance Computing
VMM	Virtual Machine Monitor
DNS	Domain Name System
VPS	Virtual Private Server

Chapter 1

INTRODUCTION

1.1 Introduction

Virtualization technologies are having a very predominant role nowadays, and the number of software solutions are increasing rapidly every day. Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware or software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others[1]. Virtualization technologies are the foundation of cloud computing. It is widely used in data centers to achieve server consolidation for efficient resource usage and to isolate collocated workloads on a single machine. The main benefits include hardware independence, isolation, secure user environments, and increased scalability, together with the large number of new properties optimized for different cases.

One of the reasons for adopting and deploying advanced technologies, and to build newer paradigms in this field is to keep up with the growth of data exchange and the consequent need to increase the capability of data center by means of server virtualization[2]. At the same time, the adoption of these technologies has extended to different areas, and incorporated into distinct use cases. Virtualization can be used in Cloud Environments, Network Function Virtualization and Internet of Things. The main benefits include hardware independence, isolation, secure user environments, and increased scalability, together with the large number of new properties optimized for different use cases [3].

Virtualization whether it be on the storage, system and the network has become a crucial way to improve system efficiency, reliability, availability, to reduce costs, and to provide greater flexibility. It enables us to use one physical server with the capability of delivering the performance of multiple servers. Without virtualization, organizations requiring more resources would have to pay for additional equipment. The problem would be made worst by the time required to order, get delivery and install such equipment. We can say that with virtualization, new applications will be made available within a few minutes and without any expenses. In its conceived form, it is better known as time-sharing thus is considered a method of logically dividing mainframes to allow multiple applications to run simultaneously [4].

1.2 Problem Definition

Nowadays, the underlying technology that are used in any cloud service is virtualization. The traditional hypervisor based virtualization runs on a physical server and handles creating a virtual environment on which any guest virtual machines operate. Virtual machines contain unnecessary overhead and suffer from reduced performance in most of the operations. So, the cloud providers try to reduce the overhead added by the hypervisor and try the virtualization management to be lightweight as much as possible. Recently, container-based virtualization, a lightweight way of virtualization which enables the single kernel to be shared among all virtual environments, has gained popularity. Due to the shared kernel as well as operating system libraries, an advantage of container-based solutions is that they can achieve a higher density of virtualized instances, and disk images are smaller compared to hypervisor-based solutions. The shared kernel approach has also a few disadvantages. One of drawback of containers is that Windows Os cannot be run on top of a LXC.

1.3 Objectives

The principle objective of this thesis work is to design a framework that will be able to compare the performance between hypervisors and light-weight virtualization i.e. container-based virtualization. The main objectives are given below:

- To compare the performance of hypervisors and lightweight virtualization in terms of CPU, Memory and Disk I/O
- To perform statistical analysis of the response time of Web Server on hypervisor and LXD

1.4 Outline of Thesis

The remaining part of this document is organized as follows:

Chapter 2 (Related Theory) includes the related theoretical background of the thesis.

Chapter 3 (Literature Review) includes the previous works related to this thesis.

Chapter 4 (Methodology) includes the research methodologies used in this research work. The proposed framework along with the tools used for implementation is discussed and understood in this chapter.

Chapter 5 (Results, Analysis and Comparison) includes the details of the experimentation and the corresponding results and analysis.

Chapter 6 (Conclusion) contains the conclusions of this research work.

Chapter 7 (Limitations and Recommendation) includes the limitation of both hypervisors and light weight virtualization and future recommendation

Chapter 2

RELATED THEORY

2.1 Hardware Based Virtualization

Hardware based virtualization provides a way to emulate the hardware so that each VM has an impression that it is running on its own hardware. Hardware vendors are developing new features to simplify virtualization techniques. Some examples are Intel Virtualization Technology (VT-x) and AMDs AMD-V which have new privileged instructions, and a new CPU execution mode that allows the VMM (hypervisor) to run in a new root mode. Hypervisor is a piece of software that facilitates creation and management of VMs.

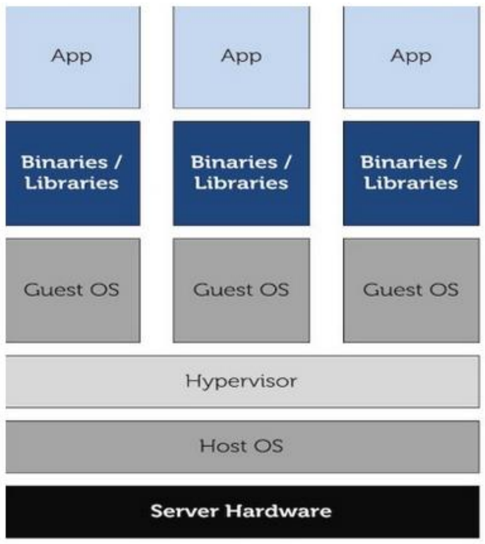


Figure 2.1: Hardware Based Virtualization

There are generally two types of hypervisor:

- **Type 1:** Native or Bare-Metal Hypervisor: This type of hypervisor runs directly on the host's hardware. Guest OS can be installed on top of this hypervisor. Such hypervisors have lesser memory footprint as compared to type 2 hypervisor. Examples of bare metal hypervisor are Citrix XenServer, VMware ESXi, and Hyper-V.

- **Type 2:** Hosted Hypervisor: This type of hypervisor requires a base OS that acts as a host. Such hypervisors abstract the presence of host from the guest OS. They may use hardware support for virtualization or can just emulate the sensitive instructions using binary translation. Examples of hosted hypervisor are VMware Workstation, VirtualBox and KVM [5].

2.2 Paravirtualization

Paravirtualization works differently than the hardware based virtualization. Paravirtualization runs directly on the hardware and provides virtualization services to the virtual machines running on it. It doesn't need to simulate the hardware for the virtual machines. The hypervisor is installed on a physical server (host) and a guest OS is installed into the environment. Virtual guests are aware that it has been virtualized, unlike the hardware based virtualization (where the guest doesn't know that it has been virtualized) to take advantage of the functions. In this virtualization method, guest source codes will be modified with sensitive information to communicate with the host. Guest Operating systems require extensions to make API calls to the hypervisor. The guests will directly communicate with the host (hypervisor) using the drivers.

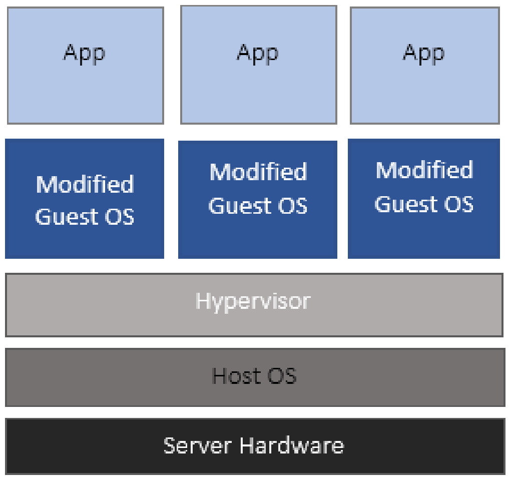


Figure 2.2: Paravirtualization

The list of products which supports paravirtualization are as follows:

- Xen
- IBM LPAR

- Oracle VM for SPARC (LDOM)
- Oracle VM for X86 (OVM)

2.3 Container-based Virtualization

Container-based virtualization also known as Operating System level virtualization shares the same kernel as that of the host and facilitates running multiple isolated user space instances. These can also be considered as light-weight virtualization. It has the best possible performance and density, and features dynamic resource management [6]. Such instances (often known as containers, virtualization engines (VE), virtual private servers (VPS) or jails) have their own resources, root file system, and run in complete isolation from the other guests. Containers do not emulate any of the underlying hardware. Instead, the virtualized OS or applications talk to the host OS, which then makes appropriate calls to the real hardware [7]. This type of virtualization usually imposes little to no overhead because programs in virtual partitions use the OS's normal system call interface and do not emulate hardware.

2.3.1 LXC

Containers are a lightweight virtualization technology. They are more a kind to an enhanced chroot than to full virtualization like Qemu or VMware, both because they do not emulate hardware and because containers share the same operating system as the host. There are two user-space implementations of containers, each exploiting the same kernel features. Libvirt allows the use of containers through the LXC driver. This can be very convenient as it supports the same usage as its other drivers. The other implementation, called simply 'LXC', is not compatible with libvirt, but is more flexible with more userspace tools.

2.3.1.1 Namespaces

The namespaces feature is accessed by the clone() system call [8], allows creating separate instances of previously-global namespaces. Linux implements filesystem, PID, network, user, IPC and hostname namespaces. Namespaces can be used in many different ways, but the most common approach is to create an isolated container that has no visibility or access to objects outside the container.

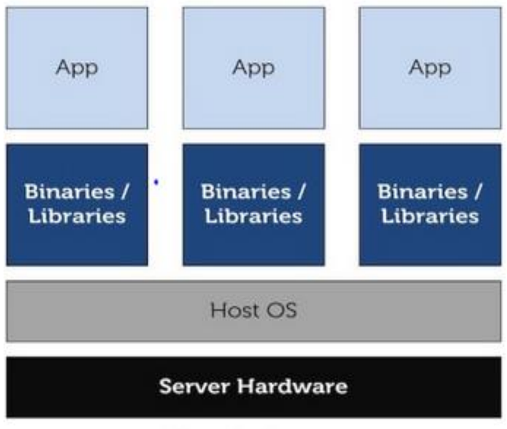


Figure 2.3: Containers Based Virtualization

2.3.1.2 Control groups

The Linux Control Groups (cgroups) subsystem [9] is used to group processes and manage their aggregate resource consumption. It is commonly used to limit memory and CPU consumption of containers. A container can be resized by simply changing the parameters of its corresponding cgroups configuration. Cgroups also provides a reliable way of terminating all processes inside a container. Because a containerized Linux system has only one kernel and the kernel has full visibility into the containers thus there is only one level of resource allocation and scheduling [7].

2.3.1.3 Cloning

Clones are either snapshots or copies of another container. A copy is a new container copied from the original, and takes as much space on the host as the original. A snapshot exploits the underlying backing store's snapshotting ability to make a copy-on-write container referencing the first. Snapshots can be created from btrfs, LVM, zfs, and directory backed containers.

2.3.2 LXD

LXD is a next generation system container manager built around a very powerful REST and offers a user experience similar to virtual machines but using Linux containers instead. LXD uses pre-made Linux images which are available for a wide number of Linux distributions LXD is based on liblxc, its purpose is to control some LXC with added capabilities, like snapshots or live migration. LXD is linked to LXC and they are OS centered.

There are many features of the LXD which are given below:

- Secure by design (unprivileged containers, resource restrictions)
- Scalable (from containers on thousands of compute nodes)
- Intuitive (simple, clear API and crisp command line experience)
- Image-based (with a wide variety of Linux distributions published daily)
- Support for Cross-host container and image transfer (including live migration with CRIU)
- Advanced resource control (CPU, memory, network I/O, block I/O, disk usage and kernel resources)
- Device passthrough (USB, Unix character and block devices, NICs, disks, and paths)
- Network management (bridge creation and configuration, cross-host tunnels)
- Storage management (support for multiple storage backends, storage pools, and storage volumes) [10]

2.3.3. Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. Docker is a piece of software that allows an administrator to pack an application into a container with all its dependencies [11]. The Docker daemon, called dockerd, is a persistent process that manages Docker containers and handles container objects.

2.3.4 Solaris Containers

Solaris containers with resource controls are known as zones [12]. There are two types of zones: global and non-global zones. The traditional view of a Solaris OS is a global zone with process ID 0. It is the default zone and is used for system wide configuration. Each non-global zone shares the same kernel as that of the global zone. Non-global zones have their own process ID, file system, network namespace, are unaware of the existence of other zones; can have their own time zone setting and boot environments. Non-global zones are thus analogous to LXC.

2.3.5 FreeBSD Jails

Jails are a chroot extension and provide OS-level virtualization on FreeBSD. Jails have Kernel tunable (sysctl) parameters that can limit the actions of the root user of that jail. Each Jail in FreeBSD has a directory sub-tree, a hostname and an IP address. The root account of a jail is not allowed to perform operations outside of the associated jail environment. Thus, there are various OS-level virtualization techniques on different OS that are more or less extension to chroot environments [13].

2.4 Apache Web Server

Web server is the software that receives the request to access a web page. It runs a few security checks on HTTP request and process to the web page. Apache is the most widely used web server software. Developed and maintained by Apache Software Foundation, Apache is an open source software available for free. It is fast, reliable, and secure. It can be highly customized to meet the needs of many different environments by using extensions and modules.

2.5 Bonnie++

Bonnie++ is a benchmark utility designed to test performance of filesystems by simulating various types of disk I/O. Bonnie++ may be used to test local disks as well as network-mounted filesystems. It is commonly available in Linux repositories. The tests should be performed on datasets larger than the amount of RAM of test system.
Bonnie++ benchmarks three things: data read and write speed, number of seeks that can be performed per second, and number of file metadata operations that can be performed per second. Metadata operations include file creation and deletion as well as getting metadata such as the file size[14].

2.6 Sysbench

Sysbench is an open source benchmarking tool that is used for evaluation of parameters like CPU and memory performance. It provides benchmarking capabilities for Linux. Sysbench supports testing even MySQL benchmarking. The idea of this benchmark suite is to quickly get an impression about system performance without setting up complex database benchmarks or even without installing a database at all. The design is very simple. SysBench runs a specified number of threads and they all execute requests in parallel. The actual workload produced by requests depends on the specified test mode[15].

Chapter 3
LITERATURE REVIEW

Several approaches have been introduced related to comparison between hypervisors and container-based virtualization:

Felter et al. measure CPU, memory throughput, storage and networking performance of Docker and KVM. This study reports KVM's start-up time to be 50 times slower than Docker. This study concludes that containers result in equal or better performance than VM in almost all cases and recommend containers for IaaS [7].

Regola and Ducom have done a similar study of the applicability of containers for HPC environments, such as OpenMP and MPI. They conclude that I/O performance of VMs is the limiting factor for the adoption of virtualization technology and that only containers can offer near-native CPU and I/O performance [16].

Xavier et al. compare the performance isolation of Xen to container implementations including Linux VServer, OpenVZ and Linux Containers (LXC). This study concluded that, except for CPU isolation, Xen's performance isolation is considerably better than any of the container implementations.. [17].

Soltesz et al. measure relative scalability of Linux-VServer (another container technology) and Xen. They show that the Linux-VServer performs better than VMs for server-type workloads and scale further while preserving performance. However, this study only measures aggregate throughput, and not the impact on a single guest common scenario in a multi-tenant environment for achieving Quality of Service for each tenant [8].

Hwang et al. compared four hypervisors (Hyper-V, KVM, vSphere and Xen) in different use cases. In summary, no superior hypervisor between those examined was discovered. Consequently, they propose that a cloud environment should support different software and hardware platforms to meet particular needs. [18].

Elisayed et al. conduct a quantitative and qualitative evaluation of VMware ESXi5, Microsoft Hyper-V2008R2, and Citrix Xen Server 6.0.2 in various scenarios. They perform an evaluation using customized SQL instances, which simulates approximately 20 million of customers, 100,000 products, and 100,000 orders per month [19].

Li et al. measure a commercial (unspecified) hypervisor, Xen and KVM using Hadoop and MapReduce as the use cases. Also here, the authors find similarities and significant variations in terms of performance with different workloads [20].

Estrada et al. benchmark KVM, Xen, and Linux containers (LXC), and compare the runtime of each environment to the performance of a physical server. This work is particularly interesting because of the application domain. The evaluation is based on sequence alignment software that arranges sequences of DNA [21].

Che et al. [22] showed a comparison between OpenVZ, Xen, and KVM monitors. The study used several benchmarks to measure the processor, memory, disk, network, and server applications. They concluded that processor virtualization is not the performance bottleneck of already consolidated virtual machine monitors, but hardware page table fault, interrupt request and I/O are the virtualization performance bottleneck.

Beserra et al. [23] analyzed the LXC container performance compared to hypervisor-based virtualization KVM for HPC activities. Two requirements were addressed: CPU and communication performance (network and inter-process), and the results showed that the type of hypervisor directly influences the results. They concluded that LXC is more suitable for HPC than KVM, however, in a more complex scenario, where physical resources are divided between multiples and logical environments, both decrease their performance, especially KVM.

Cherian. et al. tested performance on three virtualization techniques: Xen for paravirtualization, OpenVZ for containerization, and Xen Server for full virtualization. They have demonstrated that full virtualization has greater performance in file copy, pipe based context switching, process creation, shell scripts and floating point. OpenVZ showed better performance than paravirtualization in some tests [24].

Jaikar et al. analyzed the performance of scientific work flow in two virtualization technologies of virtual machines: OpenVZ and OpenStack. The work showed that containers offer better and stable performance than VMs. For this, they used the benchmark to generate an intensive workload to perform CPU, memory and I/O. [25]

CHAPTER 4
METHODOLOGY

This chapter describes the concept behind the methods used in this thesis. The flow diagram is as follows:

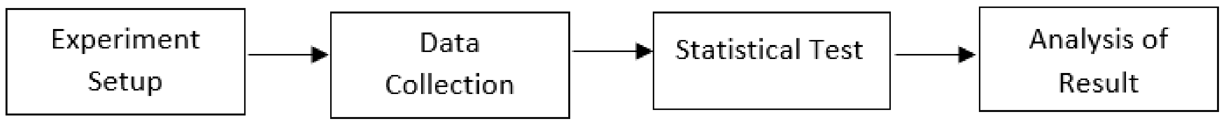


Figure 4.1 Flow diagram for the research

4.1 System Configurations

The experiment is done on host machine that has 4GB of main memory and Intel(R) Core(TM) i3-3217U CPU @ 1.80GHZ processer. CentOS 7.6.1810 (core) 64-bit operating system is running on host machine with kernel version 3.10.0-957.10.1.el7.x86_64. All of the virtual environment will share same kernel. And this host will include both KVM and LXC. For containers, there are restrictions on CPU and RAM consumptions since the host machine is shared among all virtual environments.

4.2 Data Collection

Benchmarking tool such as Sysbench is used to collect data for CPU and Memory Performance Likewise, bonnie++ is used for Disk I/O performance. The data of response time of the Web server is achieved through the Algorithm as shown in figure 4.2 which is run in the host machine**.**

4.3 R Software

All of the statistical analysis is done using the program R. R is an open source software environment used to perform statistical computation and graphics. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand. Its initial version was first released in 1995. It is supported by R Foundation for Statistical Computing. The interface in R is the command line user interface. R is cross platform compatible. R is available across Windows, Linux and macOS platforms.

R includes:

1. A simple and effective programming language which allows conditional operation, loops, user defined functions and input and output facilities.
2. A graphical facility to generate plots.
3. Tools for Data Analysis
4. Operators for calculation of Matrices and arrays

Most of the functions for R are written in R. In case of the computation intensive tasks C, C++ codes can also be linked and called during the run time. R is easily extensible through inclusion of packages.

5.3.1 R Studio

It is an integrated development environment for R. It is a GUI where one can write his/her code, see the results and also the variables that are generated on the process of executing the code. R studio is available as a commercial or open source software. R studio is available both as server version and Desktop version. R Studio is available for Windows, Linux and macOS platforms.

4.4 KVM as a hypervisor

Kernel-based Virtual Machine (KVM) is an open source virtualization technology built into Linux [26]. Specifically, KVM turns Linux into a hypervisor that allows a host machine to run multiple, isolated virtual environments called guests or virtual machines (VMs). First of all we need to make sure that our system has the hardware virtualization extensions: For Intel-based hosts, to verify, the CPU virtualization extension **(**vmx**)** are available using following command:

```
grep -e 'vmx' /proc/cpuinfo
```

Then installing qemu-kvm and qemu-img packages at first. These packages provide the user-level KVM and disk image manager.

```
yum install qemu-kvm qemu-img
```

Then following tools are installed:

```
yum install virt-manager libvirt libvirt-python libvirt-client
```

The features of these tools are:

- virt-manager provides a GUI tool to administrate virtual machines
- libvirt-client provides a CL tool to administrate virtual environment this tool called virsh
- virt-install provides the command “virt-install” to create virtual machines from CLI
- libvirt provides the server and host side libraries for interacting with hypervisors and host systems

At last, we need to start the daemon libvirtd and we can access the KVM using command virt-manager and create the virtual machine according to our requirements.

4.5 Container-based Virtualization using LXC/LXD

LXC (Linux Containers) is an OS-level virtualization method for running multiple isolated Linux systems on a single host**.** In order to install LXC, following process are considered.

Installing LXC Virtualization in Linux:

```
yum install epel-release
yum install debootstrap perl libvirt
```

Finally, installing LXC virtualization solution with the following command:

```
yum install lxc lxc-templates
```

After LXC service has been installed, verify if LXC and libvirt daemon is running:

```
systemctl status lxc.service
systemctl start lxc.service
systemctl start libvirtd
systemctl status lxc.service
```

Check LXC kernel virtualization status by issuing the below command:

```
lxc-checkconfig
```

Similarly, the process of creating a LXC container is very simple. The command syntax to create a new container is:

```
lxc-create -n container_name -t container_template
```

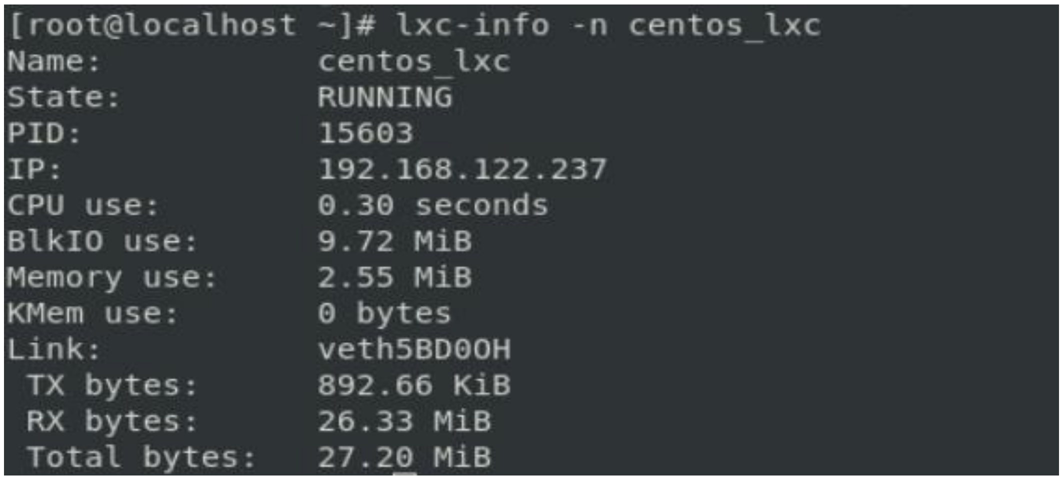
Then we can start the new container as:

```
lxc-start -n centos_lxc
```

After the installation and the start of the LXC, we can get the info of the installed LXC as:

```
lxc-info -n centos_lxc
```

This gives the information such as running OS, ipaddress, CPU usage, Memory usage and others as shown in Figure 4.2



```
[root@localhost ~]# lxc-info -n centos_lxc
Name:           centos_lxc
State:          RUNNING
PID:            15603
IP:             192.168.122.237
CPU use:        0.30 seconds
BlkIO use:      9.72 MiB
Memory use:     2.55 MiB
KMem use:       0 bytes
Link:           veth5BD0OH
 TX bytes:      892.66 KiB
 RX bytes:      26.33 MiB
 Total bytes:   27.20 MiB
```

Figure 4.2: LXC information about the CentOS container

In order to login to the container console, the lxc-console command is issued against a running container name. Login can be done by the user root and the password generated by default by LXC supervisor.

The list of containers installed can be issued from the command lxc-ls. Two LXC container centoOS_lxc under CentOS and ubuntu_lxc under Ubuntu have been configured and can be used as an isolated environment for the specific purposes such as DNS server, Web Server, Mail Server, etc.

Similarly, snapd is installed which will provide snap command line tool that is used to install LXD. Then, configure the CentOS Linux kernel for LXD. Here, grubby command is used. It is a command line tool for updating and displaying information about the configuration files for various architecture specific bootloaders:

```
grubby --args="user_namespace.enable=1" --update-kernel="$(grubby
--default-kernel)"
grubby --args="namespace.unpriv_enable=1" --update-
kernel="$(grubby --default-kernel)"
sh -c 'echo "user.max_user_namespaces=3883" > /etc/sysctl.d/99-
userns.conf'
```

Then reboot the system.

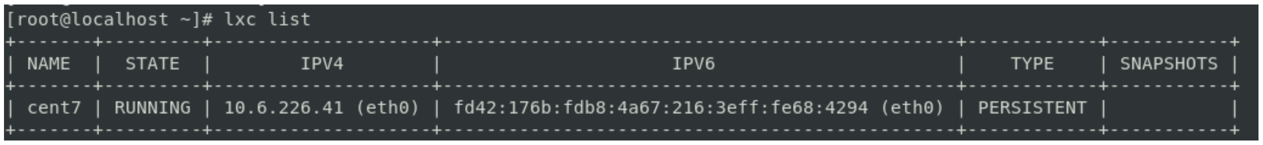
LXD is installed from snapd:

```
snap install lxd
```

Creating a test CentOS 7 container by running:

```
lxc launch images:centos/7/amd64 cent7
```

Similarly, to list the container, we can simply type the command lxc list. Figure 4.3 gives the sample output of the list of installed container in LXD.



```
[root@localhost ~]# lxc list
+-------+---------+--------------------+----------------------------------------------+------------+-----------+
| NAME  |  STATE  |        IPV4        |                     IPV6                     |    TYPE    | SNAPSHOTS |
+-------+---------+--------------------+----------------------------------------------+------------+-----------+
| cent7 | RUNNING | 10.6.226.41 (eth0) | fd42:176b:fdb8:4a67:216:3eff:fe68:4294 (eth0) | PERSISTENT |           |
+-------+---------+--------------------+----------------------------------------------+------------+-----------+
```

Figure 4.3: List of installed container in LXD

Then, configure LXD environment by:

```
lxd_init
```

After that we can access VM/container by:

```
lxc exec cent7 bash
```

The installation process is based on the guide from the LXD guide [10].

4.5.1 Access control in LXD

Access control for LXD is based on group membership. The root user as well as members of the "lxd" group can interact with the local daemon. If the "lxd" group is missing in the system then the group is created and then LXD daemon is restarted . Anyone added to this group will have full control over LXD.

Because group membership is normally only applied at login, one may need to either re-open user session or "newgrp lxd" command in the shell is needed.

4.6 Response time calculation of Web Server

Apache web server was installed on both KVM and LXD. LXD isn't a rewrite of LXC, in fact it's building on top of LXC to provide a new, better user experience And the required data was collected i.e. the HTTP reply from both the machine.

Response time can be measured from the algorithm shown below. The Python script is generated from this algorithm which is available in Appendix B2.

The host machine will then be running a script by putting the corresponding ipaddress to see how long it takes for the installed web server to respond for both LXD and KVM. The collected data will be the number of seconds until a HTTP reply is received.

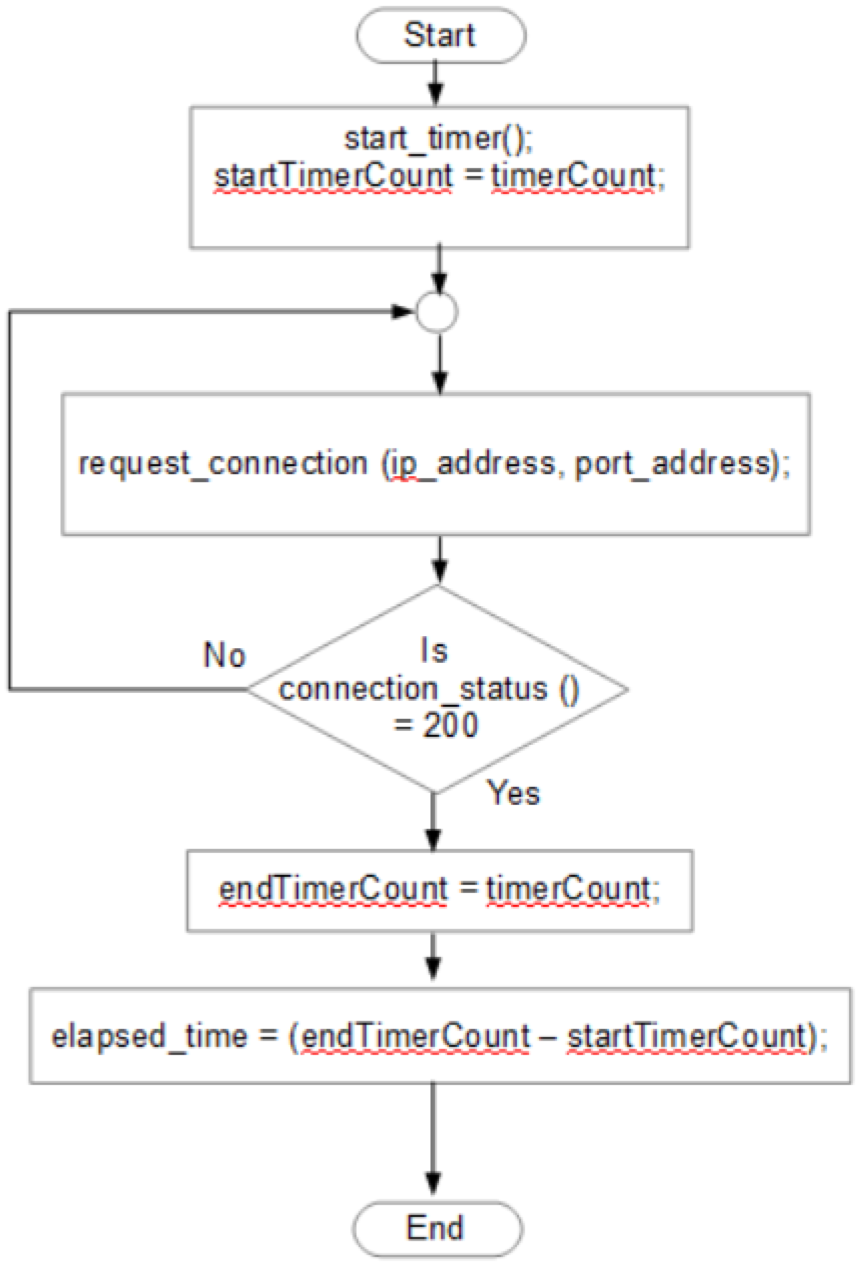

Start
start_timer();
startTimerCount = timerCount;
request_connection (ip_address, port_address);
No
Is connection_status () = 200
Yes
endTimerCount = timerCount;
elapsed_time = (endTimerCount – startTimerCount);
End

Figure 4.4 Algorithm for response time calculation of Web Server

The script, while running, tries to continuously connect to the web server that is initialized. It does this by looking for the status code 200, which indicates that the web server is up and running and that everything is working correctly. When a status code 200 is received, it will break out of the infinite loop. Error handling was added with try and except to handle replies that states that the socket is closed when the container or virtual machine is being initialized. Using the function python test.py, it will print out a real value that calculates how long the script used to finish executing and gives the response time and the sample for the statistical analysis can be generated.

4.7 Versions of software

The software and versions are listed in Table 1.

Software	**Version**
KVM	QEMU emulator version 1.5.3
LXC/LXD	3.11
Apache	2.4.6
R	3.5.3

Table 4.1 Software and their Versions

CHAPTER 5
RESULTS, ANALYSIS AND COMPARISON

This section presents the result and analysis after the comparison. As mentioned earlier, there are different benchmarks for measuring CPU, Memory and Disk I/O. Likewise, the python script has been constructed to see how long does it takes for the installed web server to respond for both LXD and KVM. All of the statistical analysis is done using the program R.
The statistical analysis and tools that will be used are sample mean and standard deviation.
The sample mean is calculated by:

$$\bar{x} = \frac{x_1 + x_2 + x_3}{n}$$

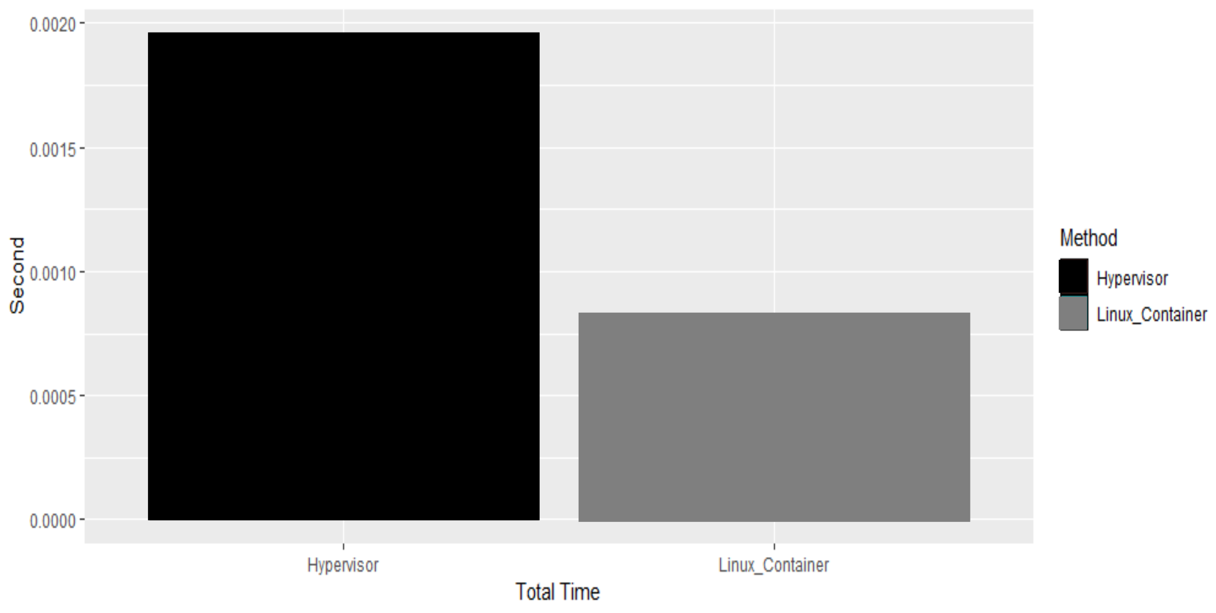
where x_2 are the individual results from the sample and N is the number of repetitions of the experiment. When the data set is normally distributed, the mean becomes interesting, because a majority of the values in the experiment should be close to the mean value.
The standard deviation of the sample is being calculated with the formula:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Where N is the number of results, x is a specific result and $\bar{x}$ is the mean of the sample. Standard deviation is a statistical values that gives insight in terms of how spread out the data points are.

5.1 CPU Performance

Sysbench[15] software has been used to measure the CPU performance. When running with the CPU workload, Sysbench will verify prime numbers by doing standard division of the number by all numbers between 2 and the square root of the number. If any number gives a remainder of 0, the next number is calculated.
The benchmark can be configured with the number of simultaneous threads and the maximum number to verify if it is a prime.



.

Figure 5.1: Results of Sysbench over 30 runs where the bars indicate the Standard Deviation

Figure 5.1 shows the result of Sysbench benchmark. Sysbench is run over 30 times and the Standard Deviation is calculated from these 30 sample for both environment. The bars indicate the Standard Deviation. Lesser the Standard deviation, better the performance. As we can see, Container based solution performs better than hypervisor.

5.2 Disk I/O

Bonnie++[14] is a benchmark software that is used to measure disk I/O in the evaluated systems. As recommended by the bonnie++ developer, the file size should be greater than the size of system memory, so, the file size of 8GB has been tested on both Hypervisor i.e. KVM and LXC.

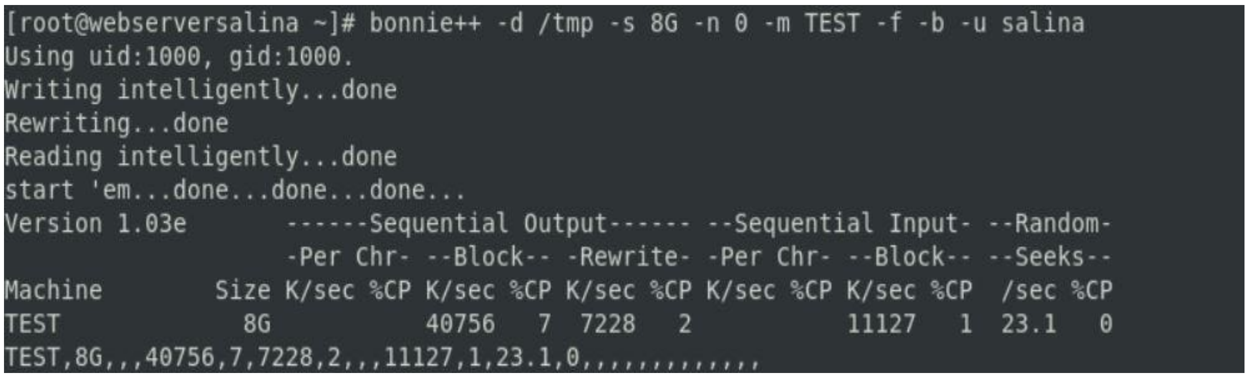
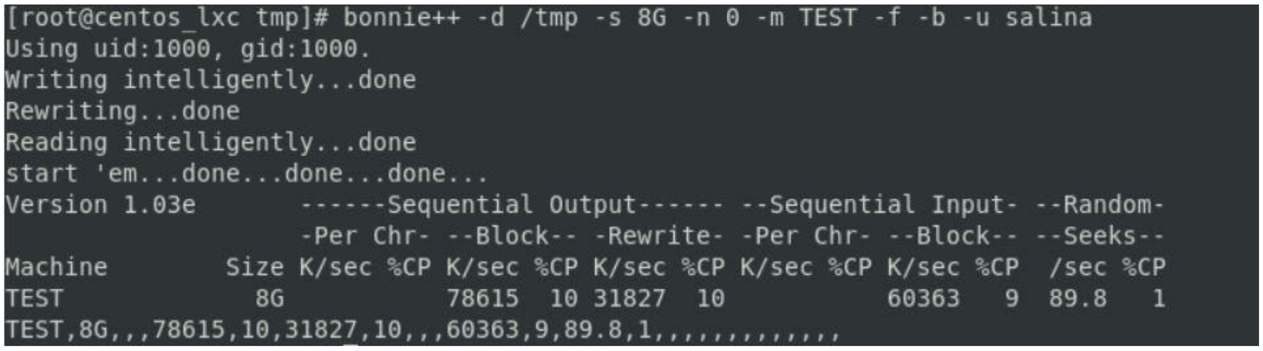

```
[root@webserversalina ~]# bonnie++ -d /tmp -s 8G -n 0 -m TEST -f -b -u salina
Using uid:1000, gid:1000.
Writing intelligently...done
Rewriting...done
Reading intelligently...done
start 'em...done...done...done...
Version 1.03e       ------Sequential Output------ --Sequential Input- --Random-
                    -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Machine        Size K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP  /sec %CP
TEST             8G           40756   7  7228   2           11127   1  23.1   0
TEST,8G,,,40756,7,7228,2,,,11127,1,23.1,0,,,,,,,,,,,,
```

Figure 5.2 : Results of Sequential write(Block Output) and Sequential read(Block Input) of KVM



```
[root@centos_lxc tmp]# bonnie++ -d /tmp -s 8G -n 0 -m TEST -f -b -u salina
Using uid:1000, gid:1000.
Writing intelligently...done
Rewriting...done
Reading intelligently...done
start 'em...done...done...done...
Version 1.03e       ------Sequential Output------ --Sequential Input- --Random-
                    -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Machine        Size K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP  /sec %CP
TEST             8G           78615  10 31827  10           60363   9  89.8   1
TEST,8G,,,78615,10,31827,10,,,60363,9,89.8,1,,,,,,,,,,,,,
```

Figure 5.3: Results of Sequential write(Block Output) and Sequential read(Block Input) of LXC

The disk throughput achieved by running Bonnie++ can be viewed from the Figure 5.2 and Figure 5.3 respectively. A total of 8GB of data was written to the disk. The sequential write performance sequential rewrite and sequential read was measured for the hypervisors and Linux Containers as shown in Figure 5.2 and 5.3. Higher the disk throughput, better the performance. LXC clearly offer better performance. LXC uses the default file format for disk image and KVM uses an image format like qcow2. The easiest way to understand the results of a bonnie++ test is to run the output through the bon_csv2html utility. This perl script uses the bonnie++ results and generates a HTML page which can be opened with web browser as shown in Figure A.1 and A.2.

5.3. Memory Performance

In order to test Memory performance, again Sysbench software is used. Table 5.1 shows the comparison of KVM and LXC. Measurement is done of the write operation. Here the two things that are to be considered are total operations and transferred amount per second.

Virtualization Technology	Total Operation per second	Transferred Amount per second (MiB/sec)
KVM	114192.20	111.52
LXC	3688298.18	3601.85

Table 5.1: Memory Performance comparison between KVM and LXC

Table 5.1 shows the output of the Sysbench to measure memory throughput on write operation. The detailed output can be seen on Figure A.3. By looking at the result, we can say LXC outperforms KVM.

5.4 Response time of Apache web server

The experiment is about how long time does the web server installed in both KVM and LXD gives response i.e the HTTP reply which was received by running a python script that was run on the host machine.

In Figure 5.4, histogram 1, shows the distribution of the data generated from script for KVM.
In Figure 5.5, histogram 2, shows the distribution of data generated from script for LXC.

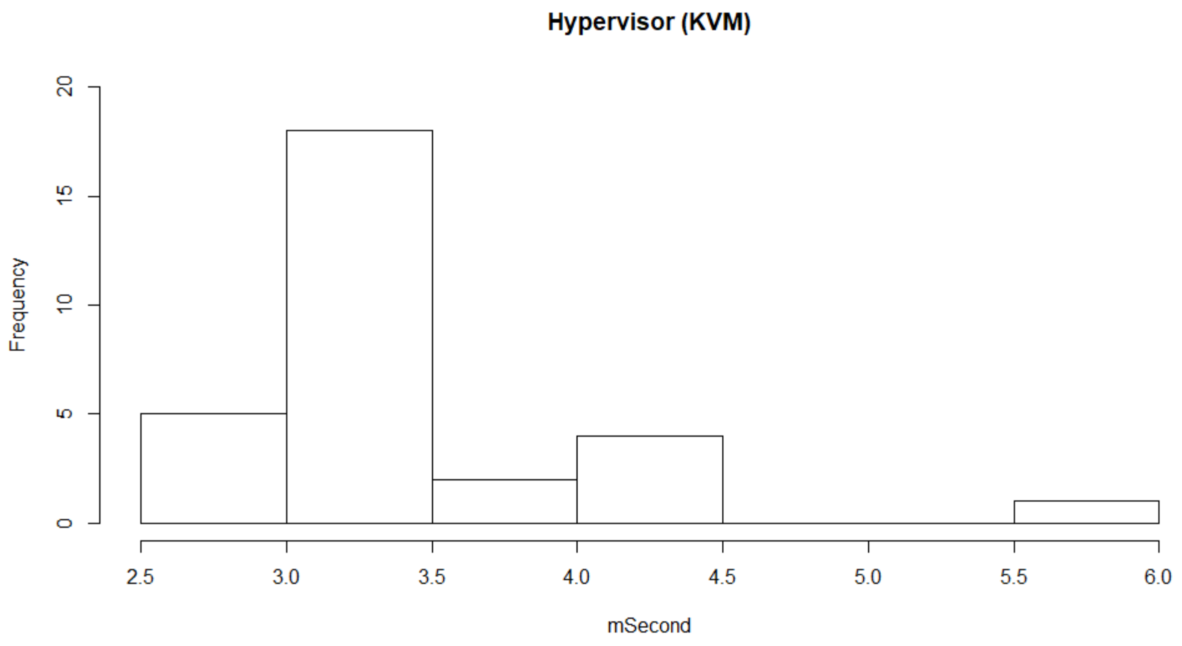


Figure 5.4: Histogram for data obtained from KVM

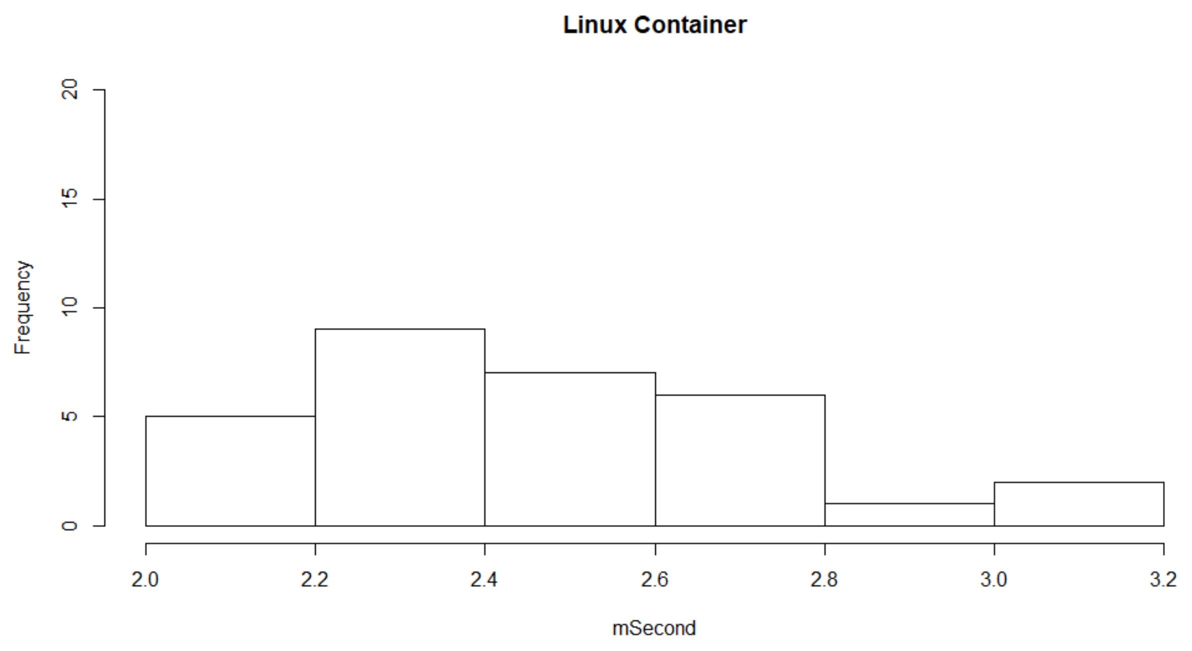


Figure 5.5: Histogram for data obtained from container

Looking at the distributions of the results, it is necessary to further investigate which kind of distribution the results lie within. As seen from histogram, it may seems to be approaching a normal distribution from its shape. However, just by looking at the histogram, it is more difficult to interpret. It should therefore be further investigated. All together to do that there is a need to know the standard deviation.

The standard deviation of the sample size 30 that was received from the python scripts explained earlier have been calculated for hypervisor and Linux container. The comparison between these have been shown in the in figure 5.6.

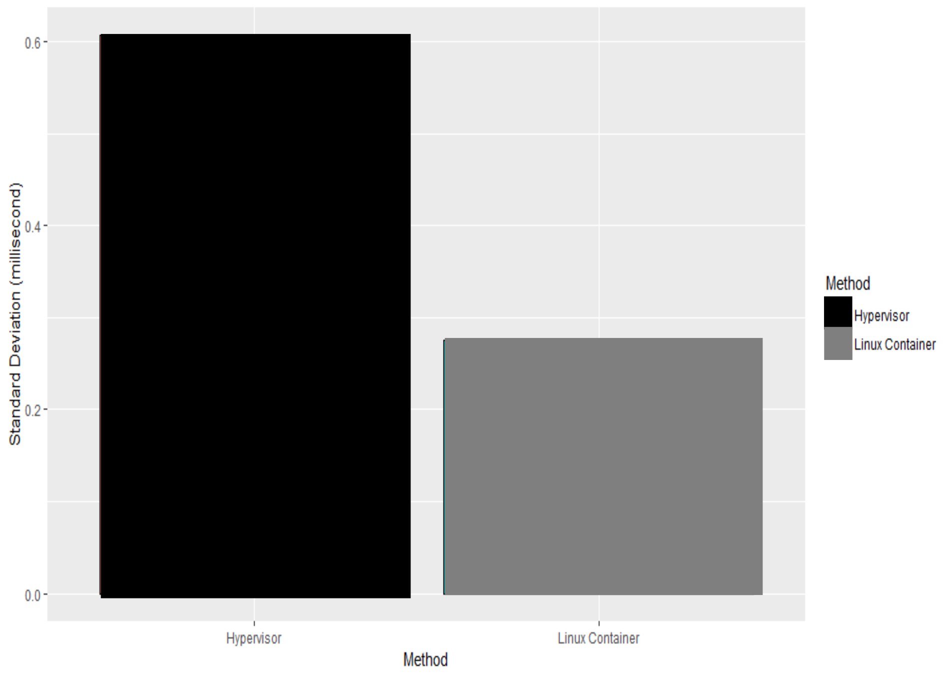


Figure 5.6: Standard Deviation of response time

As shown in the Figure 5.7, the LXD experiment has a lower standard deviation than the KVM experiment. A higher standard deviation means that there is a higher spread of the data points, which implies that the amount of seconds it takes are less consistent than with LXD.
Similarly, the mean has been calculated which is illustrated in the Figure 5.7. Again looking at the mean in Figure 5.8, provides the difference in time spent between LXD and KVM. LXD highly outperforms KVM in this experiment with the mean being more than the mean of the experiment using KVM. Containers performed faster than KVM.

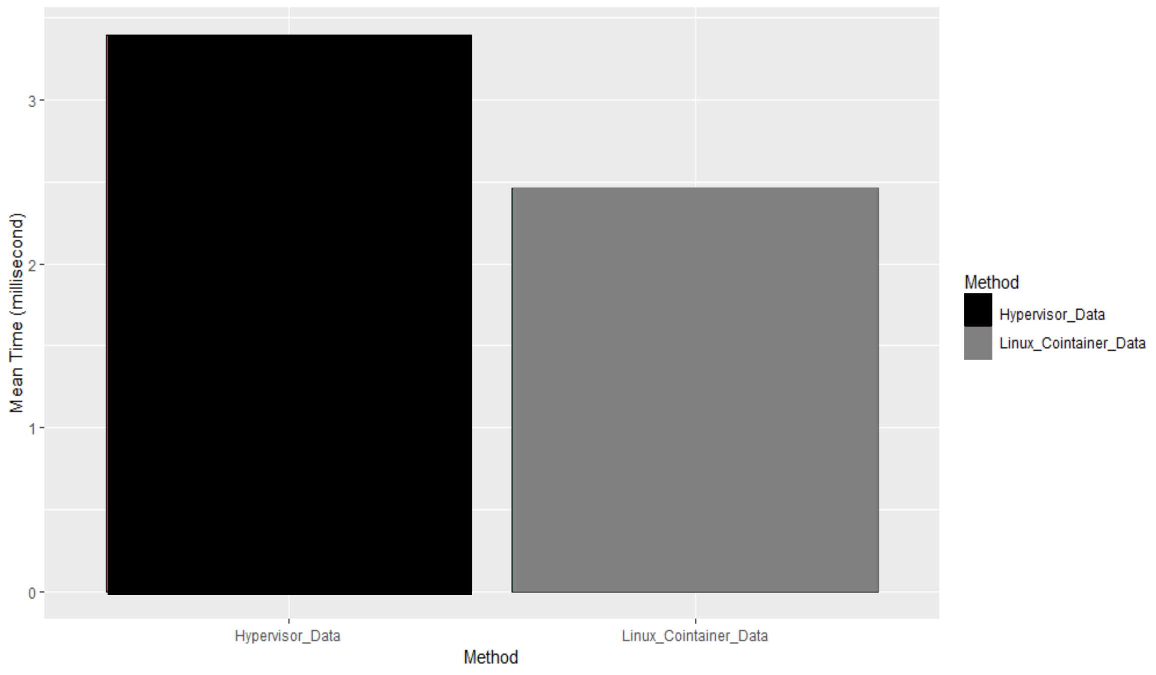


Figure 5.7: Mean of total response time

5.5 Result analysis and Discussion

This thesis has been conducted as a comparative study between one type of software that uses linux containers and one that uses virtual machines. While evaluating CPU performance, the standard deviation of LXC was 0.00082 while that of hypervisor was 0.00195. This result shows that standard deviation of LXC is less than that of KVM which shows LXC performs better than KVM in CPU performance. Similarly, while evaluating disk I/O performance, the sequential write performance was 40756 Kb/s, sequential rewrite was 7228 Kb/s and sequential read was 11127 Kb/s for KVM. Similarly, for LXC, the sequential write performance was 78615 Kb/s, sequential rewrite was 31827 Kb/s and sequential read was 60363 Kb/s. Higher the disk throughput, better the performance. Also, for the memory performance, the total operations performed by LXC was 3688298.18 per second and transferred amount was 3601.85 MiB/sec. Similarly, the total operations performed by KVM was 114192.20 per second and the transferred amount was 111.52 MiB/sec.

For the response time of Apache web server, the mean calculated from the sample for hypervisor was 3.39621 while that of LXD was 2.4628. Similarly, the standard deviation was 0.6067 for hypervisor and 0.2759 for LXD. By analyzing mean and the standard deviation of the sample, it is found that the response time of Apache web server was faster of LXD than compared to KVM. So, from the data collected and the experiment done between two different technologies, it is found that lightweight virtualization outperforms hypervisors.

CHAPTER 6

CONCLUSION

6.1 Conclusion

Container based solutions are challenging traditional virtual machines in the cloud computing. There are new advanced technologies that are more facilitating for deployment services and are lightweight as well. For this research work, the comparative study of the performance comparison between hypervisors and lightweight virtualization was done. For hypervisor, KVM and for light-weight virtualization, LXC/LXD technology was configured on CentOS. A detailed performance evaluation of CPU, Memory, Disk I/O and the web server response time of these two technologies were presented.

A comparative analysis and the corresponding results showed that the performance of Linux container is better in comparison to hypervisor in terms of CPU, Memory, Disk I/O efficiency and Web server response. Linux Containers shows interesting performance after the comparison. However, there are limitations and assumptions on certain environment such as only allowing Linux Operating System as guest OS which may limit its performance.

Chapter 7
LIMITATIONS AND RECOMMENDATION

There are few limitations for both hypervisor and Linux Container. When several virtual machines are running on the same host, performance may be hindered on the host machines. Likewise, all OS in the container should be of same version and should have same patch level of the base OS. If the base OS crashes, all the virtual container become unavailable. LXC only allows Linux as guest operating systems. So, if the Linux applications on a Windows server is needed, or vice versa, then it is only possible on traditional virtualization. Similarly, the guest containers hosted by LXD have to run the same type of operating system as the host. This limits its availability and constrains its use cases. Similarly, the security of LXC totally depends on the host system. Linux Containers scale faster in most scenarios. However, it is not possible to definitely conclude that this is true in all cases in general because only two combinations of software have been tested.

Furthermore, as for the recommendation, comparison can be done with other virtualization solutions such as Docker, Xen etc. The analysis can also be enhanced by comparing the performance in terms of security and the isolation aspects.

REFERENCES

[1] A. Singh, "An introduction to virtualization". http://www.kernelthread.com/publications/virtualization. Accessed:2019-03-12.

[2] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo, "Dynamic Resource Management Using Virtual Machine Migrations", *IEEE Communications Magazine,* September 2012.

[3] R. Morabito, J. Kjällman, M. Komu, "Hypervisors vs. Lightweight Virtualization: a Performance Comparison", *2015 IEEE International Conference on Cloud Engineering*, 2015.

[4] C. D. Graziano, "A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project" Digital Repository, Iowa State University, 2011.

[5] J. Gerald, P. R. Goldberg, Popek,. "Formal requirements for virtualizable third generation architectures" *Communications of the ACM 17.7,* 412-421, 1974.

[6] K. Kolyshkin, (2006). Virtualization in Linux. White paper, OpenVZ.

[7] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, " An updated performance comparison of virtual machines and Linux containers. *Technical Report RC25482(AUS1407-001), IBM Research Division,* 11501 Burnet Road, Austin, TX, 2014.

[8] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors*", in ACM SIGOPS Operating Systems Review*, volume 41, pages 275-287, ACM 2007.

[9] P. Menage, Control Groups. https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html. Accessed:2019-03-12.

[10] LXD. https://computingforgeeks.com/how-to-deploy-lxd-on-centos-7-with-snap Accessed:2019-03-18.

[11] M. J. Scheepers, “Virtualization and Containerization of Application Infrastructure: A Comparison”, *21st twente student conference on IT*, Vol. 1. No. 1. 2014.

[12] B. Calkins, Oracle Solaris 11 System Administration (Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2013).

[13] FreeBSD Jails. https://www.freebsd.org/doc/handbook/jails.html. Accessed: 2019-03-02.

[14] Bonnie++. https://www.linux.com/news/using-bonnie-filesystem-performance-benchmarking. Accessed: 2019-03-01.

[15] Sysbench. http://manpages.ubuntu.com/manpages/trusty/man1/sysbench.1.html. Accessed: 2019-03-04.

[16] N. Regola, J.C. Ducom, “Recommendations for virtualization technologies in high performance computing”, *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference*, pages 409–416, Nov 2010.

[17] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, , T. Lange, C. A. De Rose, “Performance evaluation of container-based virtualization for high performance computing environments”, *in Parallel, Distributed and Network-Based Processing (PDP), 21st Euromicro International Conference, IEEE*. pages 233-240, 2013.

[18] J. Hwang, , S. Zeng, T.Wood, “A component-based performance comparison of four hypervisors” *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013.

[19] E. Abdellatief, N. Abdelbaki. “Performance evaluation and comparison of the top market virtualization hypervisors” *Computer Engineering & Systems (ICCES), 2013, 8th International Conference on*. *IEEE*, 2013.

[20] J. Li, Q. Wang, D. Jayasinghe, J. Park, T. Zhu, C. Pu . “Performance Overhead Among Three Hypervisors: An Experimental Study using Hadoop Benchmarks” *Big Data (Big Data Congress), 2013 IEEE International Congress on.* IEEE, 2013.

[21] J. E. Zachary, S. Zachary, C. Pham, Z. Kalbarczyk, R. K. Iyer “A Performance Evaluation of Sequence Alignment Software in Virtualized Environments” Cluster, *Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 2014.

[22] C. Shi Che, Y. Yu, W. Lin, “A synthetical performance evaluation of openvz, xen and kvm,” *2010 IEEE Asia-Pacific Services Computing Conference*, Dec 2010, pp. 587–594.

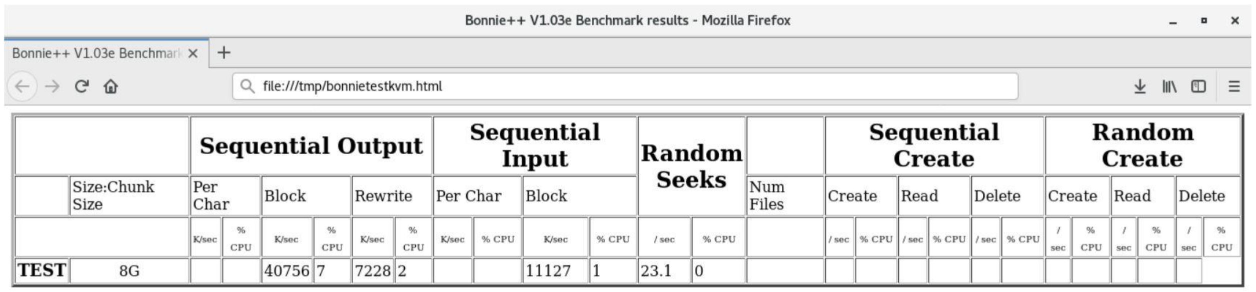
[23] D. Beserra, E. D. Moreno, P. T. Endo, J. Barreto, D. Sadok, S. Fernandes, “Performance analysis of LXC for HPC environments”, *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, pp. 358–363, July 2015.

[24] H. M. J. Babu, J. P. Martin, S. Cherian, Y. Sastri, “System performance evaluation of paravirtualization, container virtualization and full virtualization using xen, openvz, and xenserver,” *2014 Fourth International Conference on Advances in Computing and Communications*, Aug 2014, pp. 247–250.

[25] A. Jaikar, S. Shah, S. Bae, S. Noh, “Performance evaluation of scientific workflow on openstack and openvz” *Social-Informatics and Telecommunications Engineering*, LNICST, vol. 167, pp. 126–135, 2016.

[26] KVM. https://www.redhat.com/en/topics/virtualization/what-is-KVM.
Accessed:2019-03-02.

Appendix A

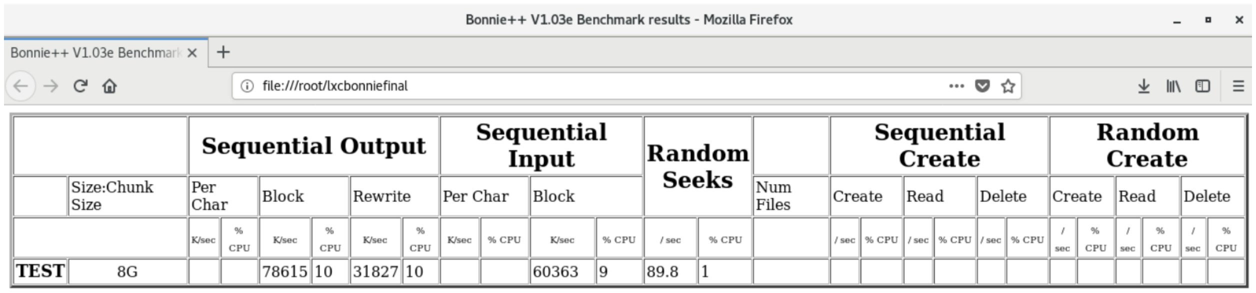


Bonnie++ V1.03e Benchmark results - Mozilla Firefox

file:///tmp/bonnietestkvm.html

		Sequential Output						Sequential Input				Random Seeks			Sequential Create						Random Create					
	Size:Chunk Size	Per Char		Block		Rewrite		Per Char		Block				Num Files	Create		Read		Delete		Create		Read		Delete	
		K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	/ sec	% CPU		/ sec	% CPU	/ sec	% CPU	/ sec	% CPU	/ sec	% CPU	/ sec	% CPU	/ sec	% CPU
TEST	8G			40756	7	7228	2			11127	1	23.1	0													

Figure A.1 : Bonnie++ results of KVM as a HTML page

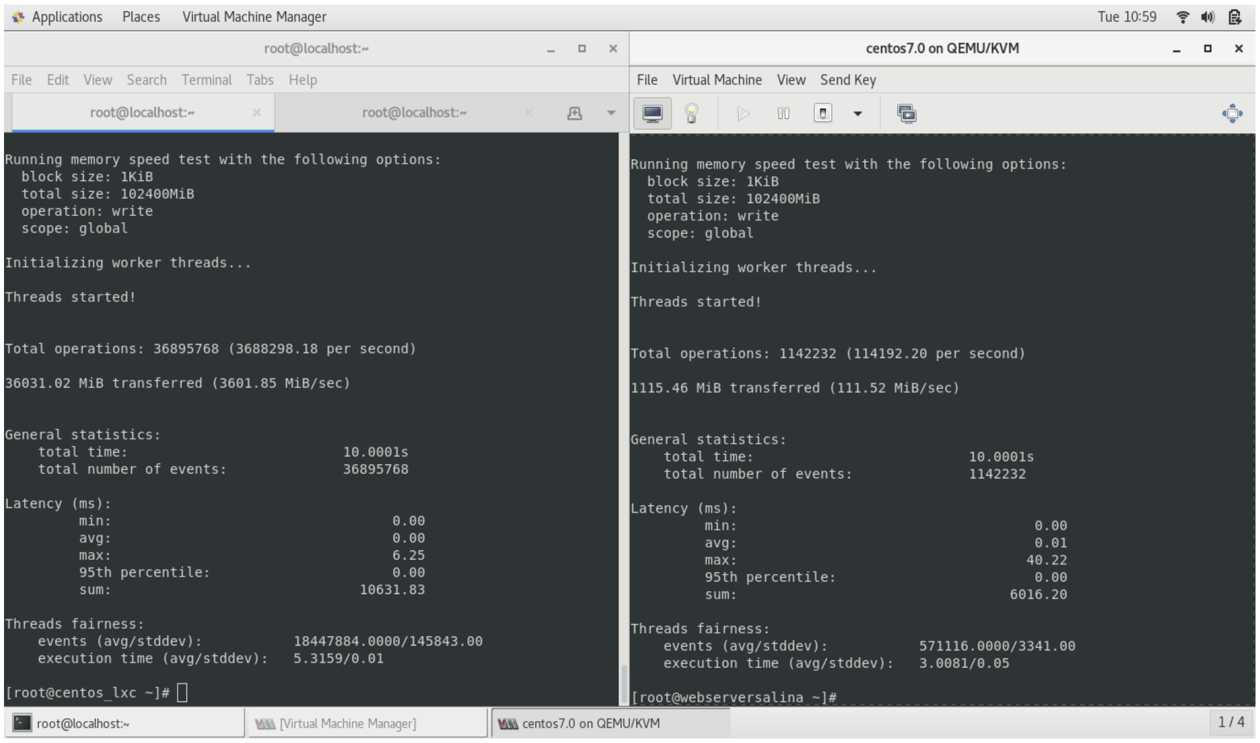


Bonnie++ V1.03e Benchmark results - Mozilla Firefox

file:///root/lxcbonniefinal

		Sequential Output						Sequential Input				Random Seeks			Sequential Create						Random Create					
	Size:Chunk Size	Per Char		Block		Rewrite		Per Char		Block				Num Files	Create		Read		Delete		Create		Read		Delete	
		K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	/ sec	% CPU		/ sec	% CPU	/ sec	% CPU	/ sec	% CPU	/ sec	% CPU	/ sec	% CPU	/ sec	% CPU
TEST	8G			78615	10	31827	10			60363	9	89.8	1													

Figure A.2: Bonnie++ results of LXC as a HTML page



```
Running memory speed test with the following options:
  block size: 1KiB
  total size: 102400MiB
  operation: write
  scope: global

Initializing worker threads...

Threads started!


Total operations: 36895768 (3688298.18 per second)

36031.02 MiB transferred (3601.85 MiB/sec)


General statistics:
    total time:                          10.0001s
    total number of events:              36895768

Latency (ms):
         min:                                  0.00
         avg:                                  0.00
         max:                                  6.25
         95th percentile:                      0.00
         sum:                              10631.83

Threads fairness:
    events (avg/stddev):           18447884.0000/145843.00
    execution time (avg/stddev):   5.3159/0.01

[root@centos_lxc ~]#
```

```
Running memory speed test with the following options:
  block size: 1KiB
  total size: 102400MiB
  operation: write
  scope: global

Initializing worker threads...

Threads started!


Total operations: 1142232 (114192.20 per second)

1115.46 MiB transferred (111.52 MiB/sec)


General statistics:
    total time:                          10.0001s
    total number of events:              1142232

Latency (ms):
         min:                                  0.00
         avg:                                  0.01
         max:                                 40.22
         95th percentile:                      0.00
         sum:                               6016.20

Threads fairness:
    events (avg/stddev):           571116.0000/3341.00
    execution time (avg/stddev):   3.0081/0.05

[root@webserversalina ~]#
```

Figure A.3: Memory Performance comparison between KVM and LXC

Appendix B

Sample Source Code

B1: R code for Statistical Analysis

```
Code to perform the Statistical Analysis of Hypervisor and Linux_Content #
# ----------------- Author Salina Shrestha 2019.03.24 -------------------  #
# -------------------------- Version 1.0 --------------------------------- #

#Install Package
#install.packages("ggplot2")

#Library include
library(ggplot2)

#Set the Working Directory
setwd("D:/Salina_Thesis_MSc_TU/")

#import file
Table1 <- read.csv(file = "D:/Salina_Thesis_MSc_TU/Response_Time_Statistical_Analysis.csv")

# Dimension of Data for Validation of Data if it is correctly loaded to the
dim(Table1)

# Attach the Database to the R search path.
# This means that the database is searched by R when evaluating a variable,
# That means the objects in the database can be accessed by simply giving their names.
attach(Table1)
```

```
#remove Data if required
#remove(dat)

#Perform Summary Statistical Analysis
summary(Table1)
mean_HyperVisor <- mean(Hypervisor_Data)
mean_Linux_Container <- mean(Linux_Container_Data)
sd_Hypervisor <- sd(Hypervisor_Data)
sd_Linux_Container <- sd(Linux_Container_Data)

# Histogram
hist(Hypervisor_Data,  main = paste("Hypervisor (KVM)" ), xlab = paste("mSecond"), ylab =
paste("Frequency"), ylim = c(0, 20),freq = T )

hist(Linux_Container_Data, main = paste("Linux Container" ), xlab = paste("mSecond"), ylab =
paste("Frequency"), ylim = c(0, 20),freq = T )

# mean Time Comparision
mean_data <- data.frame(
  Method = factor(c("Hypervisor","Linux_Container")),
  mean_time = c(mean_HyperVisor, mean_Linux_Container)
)

# Plot Mean Time for Comparision
ggplot(data = mean_data, aes(x = Method, y = mean_time, fill = Method)) +
  geom_bar(colour ="black",stat ="identity") +
  xlab("Mean Time") + ylab(" millisecond ")

# Standard Deviation Comparision
standard_deviation_data <- data.frame(
  Method = factor(c("Hypervisor","Linux Container")),
  sd_time = c(sd_Hypervisor, sd_Linux_Container)
)
ggplot(data = standard_deviation_data, aes(x = Method, y = sd_time, fill = Method)) +
  geom_bar(colour ="black",stat ="identity") +
  xlab("Standard Deviation") + ylab(" millisecond ")

#import File for Total TIme Analysis
Table2<-
read.csv(file="D:/Salina_Thesis_MSc_TU/CPU_Performance_Data_Statistical_Analysis.csv")

# Dimension of Data for Validation of Data if it is correctly loaded to the
dim(Table2)

# Attach the Database to the R search path.
# This means that the database is searched by R when evaluating a variable,
# That means the objects in the database can be accessed by simply giving their names.
attach(Table2)

#Perform Summary Statistical Analysis of Total Time
summary(Table2)
sd_Total_Time_Hypervisor <- sd(KVM_Total_Time)
sd_Total_Time_Linux_Container <- sd(Linux_Container_Total_Time)

# Standard Deviation Comparision of Total Time
sd_Total_Time_data <- data.frame(
  Method = factor(c("Hypervisor","Linux_Container")),
  sd_Total_Time = c(sd_Total_Time_Hypervisor, sd_Total_Time_Linux_Container)
)

# Plot to compare the Standard Deviation of Total Time
ggplot(data = sd_Total_Time_data, aes(x = Method, y = sd_Total_Time, fill = Method)) +
geom_bar(colour ="black",stat ="identity") + xlab("Total Time") + ylab("Second")
#import File for Total Time Analysis
Table3<-
read.csv(file="D:/Salina_Thesis_MSc_TU/Memory_Performance_Statistical_Analysis.csv")

# Dimension of Data for Validation of Data if it is correctly loaded to the
dim(Table3)

# Attach the Database to the R search path.
# This means that the database is searched by R when evaluating a variable.
# That means the objects in the database can be accessed by simply giving their names.
attach(Table3)

#Perform Summary Statistical Analysis of Total Time
summary(Table3)
mean_Memory_Performance_Hypervisor <- mean(Hypervisor_Memory_Performance)
mean_Memory_Performance_Linux_Container<-
Mean(Linux_Container_Memory_Performance)

# Standard Deviation Comparision of Total Time
mean_Memory_Performance_data <- data.frame(
Method = factor(c("Hypervisor","Linux_Container")),
mean_Memory_Performance=c(mean_Memory_Performance_Hypervisor,
mean_Memory_Performance_Linux_Container)
)

# Plot to compare the Standard Deviation of Total Time
ggplot(data=mean_Memory_Performance_data,aes(x=Method, y = mean_Memory_Performance,
fill = Method)) +
  geom_bar(colour ="black",stat ="identity") +
  xlab("Write Operation ") + ylab("Memory Throughput (MB/s)")
```

B2: Python Script to get the response time from Web Server

```
Import time
import httplib
start = time.time()
while True:
    try:
        connection=httplib.HTTPConnection("192.168.122.237", 80)
        connection.request("GET","/")
        status=connection.getresponse()
        statuscode=status.status
        if statuscode==200:
            break
    except:
        pass
end = time.time()
print(end - start)
```