Performance comparison of Linux containers (LXC) and OpenVZ during live migration

An experiment

Pavan Sutha Varma Indukuri

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden
This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full-time studies.

Contact Information:
Author(s):
Pavan Sutha Varma Indukuri
E-mail: pain15@student.bth.se

University advisor:
Sogand Shirinbab
Department of Computer Science and Engineering
E-mail: sogand.shirinbab@bth.se

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
ABSTRACT

Context. Cloud computing is one of the most widely used technologies all over the world that provides numerous products and IT services. Virtualization is one of the innovative technologies in cloud computing which has advantages of improved resource utilization and management. Live migration is an innovative feature of virtualization that allows a virtual machine or container to be transferred from one physical server to another. Live migration is a complex process which can have a significant impact on cloud computing when used by the cloud-based software.

Objectives. In this study, live migration of LXC and OpenVZ containers has been performed. Later the performance of LXC and OpenVZ has been conducted in terms of total migration time and downtime. Further CPU utilization, disk utilization and an average load of the servers is also evaluated during the process of live migration. The main aim of this research is to compare the performance of LXC and OpenVZ during live migration of containers.

Methods. A literature study has been done to gain knowledge about the process of live migration and the metrics that are required to compare the performance of LXC and OpenVZ during live migration of containers. Further an experiment has been conducted to compute and evaluate the performance metrics that have been identified in the literature study. The experiment was done to investigate and evaluate migration process for both LXC and OpenVZ. Experiments were designed and conducted based on the objectives which were to be met.

Results. The results of the experiments include the migration performance of both LXC and OpenVZ. The performance metrics identified in the literature review, total migration time and downtime, were evaluated for LXC and OpenVZ. Further graphs were plotted for the CPU utilization, disk utilization, and average load during the live migration of containers. The results were analyzed to compare the performance differences between OpenVZ and LXC during live migration of containers.

Conclusions. The conclusions that can be drawn from the experiment. LXC has shown higher utilization, thus lower performance when compared with OpenVZ. However, LXC has less migration time and downtime when compared to OpenVZ.

Keywords: OpenVZ, LXC, Live migration, Virtualization
ACKNOWLEDGMENT

I would like to express my gratitude towards my supervisor, Sogand Shirinbab for giving me some valuable suggestions which enhanced the level of my Master Thesis. She was very generous and was always in contact with me through emails even during the weekends. I would like to extend my gratitude towards my examiner, Dr. Martin Bolt for responding to all my emails with the queries about Master Thesis.

I am also thankful to IT helpdesk faculty for helping me with the network configurations of the servers. They have responded quickly whenever I had an issue. I would appreciate their patience for resolving the issue multiple times.

I am also thankful to my friends who made my stay at Sweden feel like home and more memorable. I take this opportunity to express my gratitude to my friends who have motivated me throughout my Thesis, especially Sai Venkat Naresh Kotikalapudi, Apuroop Paleti, and Haritha Reddy Gadila Swarajya without whom I wouldn’t have completed my Master Thesis on time.

I am grateful to my parents for their unconditional love and support till date. They have always believed in me and encouraged me throughout my academic career without whom it would be impossible to complete my Master’s degree.
LIST OF FIGURES

Figure 1: NIST definition of cloud computing ................................................................. 11
Figure 2: Cloud Architecture ......................................................................................... 12
Figure 3: System x86 Architecture ............................................................................... 13
Figure 4: Virtual Machine vs. Container ....................................................................... 14
Figure 5: Hypervisor types .......................................................................................... 15
Figure 6: OpenVZ architecture ([33]) .......................................................................... 16
Figure 7: Pre-copy migration timeline ([37]) .............................................................. 18
Figure 8: Pre-copy migration ....................................................................................... 19
Figure 9: Post-copy migration ...................................................................................... 19
Figure 10: Hybrid migration ......................................................................................... 19
Figure 11: Live migration process ............................................................................... 21
Figure 12: Testbed ......................................................................................................... 27
Figure 13: LXC live migration ...................................................................................... 28
Figure 14: CPU Utilization for OpenVZ (threads = 150) ............................................ 32
Figure 15: Load average for OpenVZ (threads = 150) ................................................. 32
Figure 16: Disk Utilization for OpenVZ (threads = 150) ............................................. 33
Figure 17: CPU Utilization for OpenVZ (threads = 450) ............................................. 34
Figure 18: Disk Utilization for OpenVZ (threads = 450) ............................................. 34
Figure 19: Load average for OpenVZ (threads = 450) ............................................... 35
Figure 20: CPU utilization for LXC (threads=150) ..................................................... 35
Figure 21: Load average for LXC (threads=150) ......................................................... 36
Figure 22: CPU utilization for LXC (threads=450) .................................................... 36
Figure 23: Load average for LXC (threads=450) ......................................................... 37
Figure 24: Disk utilization for LXC (threads=450) ....................................................... 37
Figure 25: Downtime for LXC and OpenVZ ................................................................. 38
Figure 26: Box plot for Downtime (Threads=150) ....................................................... 39
Figure 27: Box plot for Downtime (Threads=450) ....................................................... 39
Figure 28: Total Migration Time for OpenVZ and LXC ............................................. 39
Figure 29: Boxplot for Total Migration Time (Thread=150) ....................................... 40
Figure 30: Boxplot for Total Migration Time (Thread=450) ....................................... 40
Figure 31: CPU Utilization for LXC and OpenVZ ....................................................... 41
Figure 32: CPU utilization for LXC and OpenVZ at Destination .................................. 41
Figure 33: CPU utilization for LXC and OpenVZ at source ........................................ 41
Figure 34: Load average at source for LXC and OpenVZ ........................................... 42
Figure 35: Load average at destination for LXC and OpenVZ ..................................... 42
Figure 36: Disk utilization at source for LXC and OpenVZ ......................................... 43
Figure 37: Disk utilization at destination for LXC and OpenVZ .................................... 43
Figure 38: Load Average for OpenVZ at Source (threads = 150) ............................... 52
Figure 39: Load average for OpenVZ at destination (threads = 150) ....................... 52
Figure 40: Literature review ......................................................................................... 53
Figure 41: Results for OpenVZ (threads = 150) ............................................................. 53
Figure 42: Results for OpenVZ ( threads = 450) ........................................................... 54
Figure 43: Results for LXC ( threads = 150) ................................................................. 54
Figure 44: Results for LXC (threads = 450) ............................................................... 54
Figure 45: Configuration settings for OpenVZ .............................................................. 57
**ABBREVIATIONS**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>VE</td>
<td>Virtual Environment</td>
</tr>
<tr>
<td>VMM</td>
<td>Virtual Machine Monitor</td>
</tr>
<tr>
<td>LXC</td>
<td>Linux Container</td>
</tr>
<tr>
<td>CRIU</td>
<td>Checkpoint/Restore in Userspace</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>ALU</td>
<td>Arithmetic Logic Unit</td>
</tr>
<tr>
<td>VPS</td>
<td>Virtual Private Server</td>
</tr>
<tr>
<td>YAML</td>
<td>Yet Another Markup Language</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure shell</td>
</tr>
<tr>
<td>Sshd</td>
<td>OpenSSH Daemon</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma-separated value</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>RAM</td>
<td>Random-access memory</td>
</tr>
</tbody>
</table>
CONTENTS

ABSTRACT ..........................................................................................................................I

ACKNOWLEDGMENT ........................................................................................................II

LIST OF FIGURES ...............................................................................................................III

ABBREVIATIONS ................................................................................................................IV

CONTENTS ..........................................................................................................................V

1 INTRODUCTION ..............................................................................................................7
1.1 OVERVIEW....................................................................................................................7
1.2 PROBLEM IDENTIFICATION AND MOTIVATION .........................................................8
1.3 AIM AND OBJECTIVES .................................................................................................9
1.4 RESEARCH QUESTIONS ...............................................................................................9
1.5 CONTRIBUTIONS ........................................................................................................10
1.6 THESIS OUTLINE .......................................................................................................10

2 BACKGROUND ..............................................................................................................11
2.1 CLOUD COMPUTING ..................................................................................................11
2.2 VIRTUALIZATION .......................................................................................................12
  2.2.1 Virtualization Techniques ......................................................................................13
  2.2.2 Virtual Machine and Containers ...........................................................................14
  2.2.3 Hypervisor ............................................................................................................14
  2.2.4 Container-based Virtualization ............................................................................15
2.3 LIVE MIGRATION .......................................................................................................17
  2.3.1 OpenVZ live migration ..........................................................................................20
  2.3.2 LXC live migration ..............................................................................................20

3 RELATED WORK ...........................................................................................................22

4 METHODOLOGY ..........................................................................................................24
4.1 LITERATURE REVIEW: ..............................................................................................25
4.2 EXPERIMENT .............................................................................................................26
  4.2.1 Test Environment ....................................................................................................26
  4.2.2 Initial Setup ............................................................................................................26
  4.2.3 Migration ...............................................................................................................27
  4.2.4 Factors ..................................................................................................................29
  4.2.5 Performance Metrics ............................................................................................30

5 RESULTS .......................................................................................................................31
5.1 CASE 1: ......................................................................................................................31
  5.1.1 Scenario 1: ............................................................................................................31
  5.1.2 Scenario 2: ............................................................................................................33
5.2 CASE 2 .......................................................................................................................35
  5.2.1 Scenario 3: ............................................................................................................35
  5.2.2 Scenario 4: ............................................................................................................36

6 ANALYSIS .....................................................................................................................38

7 DISCUSSIONS AND VALIDITY THREATS .................................................................44
7.1 INDEPENDENT AND DEPENDENT VARIABLES ......................................................45
7.2 THREATS TO VALIDITY ............................................................................................45

8 CONCLUSION AND FUTURE WORK ...........................................................................47
8.1 CONCLUSIONS: ........................................................................................................... 47
  8.1.1 Answers to Research Questions .............................................................................. 47
8.2 FUTURE WORK ............................................................................................................. 48

REFERENCES ........................................................................................................................... 49

APPENDIX A ............................................................................................................................ 52

APPENDIX B ............................................................................................................................ 55
1 INTRODUCTION

1.1 Overview

Cloud computing is one of the most useful technologies that is being widely used all over the world. It has the potential to change individual’s style of working by providing IT services and products [1]. The primary objective of cloud computing is to centralize existing computing resources connected to the network and present the information as a service over the cloud. This network which is used to deliver shared resources is known as cloud [2]. Cloud computing improves the utilization of distributed resources by setting the resources together to accomplish higher throughput for handling complex computational problems [3]. Nowadays, users need to adapt to several configurations and installations brought on by complex IT bases. Thus, implementing computing as a service would be helpful for users to manage complicated IT infrastructures [4] [5]. There are various fundamental innovations, services, and base-level configurations that make cloud computing conceivable. Virtualization as one of the primary constituents of cloud computing yields numerous support for sharing and management of resources in the cloud [4].

Virtualization is an advancing technology that provides scope for benefits in computing systems, which are improved resource utilization, application isolation, portability, and system reliability [5]. Virtualization abstracts the coupling between the hardware and operating system to improve speed, adjustability, reduce costs and thus strengthen business value [3]. Virtualization enhances the utilization of resources by sharing them among the virtual machines(VMs) to run two or more operating systems simultaneously. Virtual machines can run different operating systems on the same physical system [6]. This technology inserts a virtualization layer between the hosts OS and hardware allowing the virtual machine to use the physical processor, memory, and I/O devices. The hypervisor, also known as Virtual Machine Monitor (VMM), is responsible for adding the virtualization layer and later creating and managing the virtual machines. Various VMs can run on a single host by using the hypervisor [7].

The Hypervisor-based virtualization consists of a virtual machine monitor (VMM) on top of the operating system, which provides an abstraction of all the guest operating systems [8]. Xen, VMware, and KVM are examples of hypervisor-based virtualization. An alternative approach to virtualization is container-based virtualization, in which operating system’s kernel runs on a physical system with a separate set of processes, known as Containers [9]. OpenVZ, LXC, and Dockers are based on container virtualization, also called operating system level virtualization because virtualization is at operating system level instead of hardware level as in hypervisors [8]. Live migration is a feature in virtualization where virtualized environment, Virtual machines and Containers, are transferred from one physical server to other. It is useful for maintenance of the system, and it also balances workload among the physical nodes.

Cold migration is a trivial form of migration where virtualized environment is stopped; the file system is copied to other server and later it is started on another server. Since the container is an isolated entity, its complete state can be saved and restored back from the same point. Checkpoint and restore can be used to migrate container from one server to other [10]. The Pre-copy algorithm is often used for migrating virtual machines, where hypervisor copies the memory pages from source to the target host.
Memory pages which get updated during migration process are known as dirty pages which must be iteratively transferred to the destination host. After few iterations, VM is suspended at the origin and the remaining dirty pages are copied to the target host. Later VM is resumed at the destination host [11].

The performance of live migration depends on Total Migration Time and Downtime. Migration time is the time required to transfer the dirty pages from source to destination. Downtime is the time between the suspension of migrated VM at the source and its resumption at the target node [11].

1.2 Problem identification and motivation

Cloud computing is a new and promising paradigm delivering IT services as computing utilities. Software professionals are facing diverse challenges while constructing a software for many consumers to use as a service instead of running it on their personal systems[12]. High availability of the services provided by the cloud must be considered by the virtual private server (VPS) providers and cloud providers. The major bottleneck to the data centers is downtime due to hardware failures that must be mitigated to avoid massive losses to the company. Live migration is used in this scenario to reduce the downtime by migrating the applications or processes on failure node to a stable server to ensure high availability of services. Though live migration reduces the downtime to a major extent it can be can be further optimized by choosing a suitable type of virtualized systems.

Various research works [13], [14] are done to minimize the downtime of the server by using certain algorithms stated by the authors. Few research works [7], [15], [16] include a comparison between different types of Virtual machines and Containers. The performance of OpenVZ is compared with other hypervisors in [17]–[22] based on network throughput, live migration, and router virtualization. But there is no research done for comparing the performance of live migration in OpenVZ and LXC, which is the latest and developing technology in the field of virtualization. So, the present research work would fill the gap in existing literature of Container live migration. Containers are widely used over hypervisors by application developers due to its performance and scalability [7] [15]. They can be run on the same operating system, unlike virtual machines which require the installation of OS inside each VM. They are lightweight and the creation of new containers is very fast. Containers scalability feature allows it to scale up and down according to the requirement and its resource requirements are lesser as compared to virtual machines as they do not use full operating system [7].

The characteristic implementations of container-based virtualization are OpenVZ and LXC, which are Linux implementations and have few similarities. They differ by the way in which the resource handling is done between different containers on a particular physical host and how the isolation of the resources is achieved [18]. So, OpenVZ and LXC are selected for this research study as they are actively developing and efforts are made to merge these tools [23]. OpenVZ is secure and stable and well maintained with consistent releases while LXC is the latest technology which is built on top of the recent kernel and have some new features [24].
1.3 Aim and Objectives

The main aim of the research is to perform live migration of Linux container (LXC) and OpenVZ containers. Later, compare the performance of LXC and OpenVZ during live migration of containers based on Total Migration Time and Downtime using Cassandra NoSQL database.

To achieve this aim, following objectives have to be fulfilled:

- Perform live migration of Containers running Cassandra from one physical server to another in LXC and OpenVZ.
- Evaluate the performance of live migration based on Total Migration Time and Downtime.
- Analyze the impact on CPU utilization, Disk utilization, and the Load average of the servers which are involved in the process of migration.
- Compare the results using graphs to illustrate quantitative data.

1.4 Research questions

Research questions are formulated to fulfill the objectives that lead to achieving the aim of the project.

**RQ1:** How is live migration performed in OpenVZ and LXC containers?

**Motivation:** This study provides details about the procedure, precautions, possible errors and their mitigations well in advance before conducting an experiment. Detailed study of the existing literature about live migration in containers would result in the smooth flow of experiment.

**RQ2:** What is the difference in performance between OpenVZ and LXC during Live migration in terms of CPU utilization, Disk utilization, Downtime, and Total Migration Time?

**Motivation:** Performance measure of the containers would result in quantitative data which can be compared to the containers. These results can be used to find a container with a better performance which has an acceptable tradeoff between the migration time and downtime. Migration affects the CPU and disk utilization of the servers which must also be considered for detailed comparison.

1.5 Contributions

The present research work contributes to the future researchers in this field to perform live migration of containers without any difficulties. It provides detailed information about the complicated process of live migration. The procedure involved to live migrate the containers in OpenVZ and LXC along with their basic requirements for performing live migration are also mentioned. Further, the essential configuration settings for the containers in both the hypervisors is explained, which are required to provide network access to the containers. This network access would be helpful for increasing the load of the container and maintaining connection during the process of migration.

Further, the present study would also contribute the VPS and cloud providers to select an appropriate Hypervisor according to their requirement for ensuring high availability.
of the services provided to the customer. From the conclusions drawn in this study, service providers would opt for OpenVZ when the CPU load is high because it was concluded from the experiment that it could handle higher load efficiently than LXC. For higher load, LXC sometimes fails to migrate the container that leads to repetitive migration process. Data centers would prefer LXC as it is the latest technology and has lesser downtime and migration time when compared to OpenVZ. However, the cloud providers can opt for OpenVZ if CPU utilization and Disk utilization are taken into consideration, which are low in OpenVZ when compared to LXC.

1.6 Thesis Outline

Chapter 1 gives a brief introduction to the research work. Chapter 2 covers background information about the technologies, Cloud Computing, Virtualization, Hypervisors, Live migration and the tools, required for an in-depth understanding of the present research work. Chapter 3 gives a brief overview of other studies conducted in the same research area along with their contributions Chapter 4, literature review and experiment methods used were discussed. Results obtained from the experiment are reported in Chapter 5 and the results are analyzed in Chapter 6. Discussions about the results and validity threats are stated in Chapter 7. The conclusions of the research work and possible areas for future work are mentioned in Chapter 8 followed by references and appendix.
2 BACKGROUND

2.1 Cloud Computing

Cloud Computing is a new paradigm for the provision of computing infrastructure. The National Institute of Standards and Technology (NIST) defines Cloud Computing as “model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction”[25]. The cloud computing definition of NIST consists of five essential features, three service models and four deployment models as shown in Figure 1. The primary objective of cloud computing is to centralize existing computing resources and deliver the services to the devices over the network. The network used to provide shared resources is called the cloud. Clouds are a large group of virtualized resources which can be altered to various load, and thus allows optimum resource utilization. Cloud computing achieves higher throughput by efficiently using the distributed resources.

![Figure 1: NIST definition of cloud computing](image)

The five essential features provided by cloud computing are:

- On-demand self-service: Consumer can provide computing capabilities automatically whenever needed without the involvement of humans.
- Broad Network access: Client platforms can access all the capabilities provided by the system over the network through standard mechanisms.
- Resource pooling: Computing resources are pooled to different consumers with a virtualized resource that can be assigned as per consumer demand.
- Rapid elasticity: Capabilities can be provided and released as per consumer demand. The user can release nonessential resources if the work has been done.
- Measured service: Resource usage can be observed, restrained and recorded to maintain clarity for both consumers and producers of a particular service.
The design for software applications is a layered architecture of cloud computing as shown in Figure 2. Cloud architecture is often used to obtain the necessary resources to perform a particular work and later discard the non-essential resources after the work is done. The services can be accessed only through the cloud from anywhere in the world. Cloud models can be distinguished into three types:

- **Private Cloud**: The cloud environment where data and processes are managed within the organization.

- **Public Cloud**: The cloud environment whereby resources are provided over the network from a third-party provider.

- **Hybrid Cloud**: The environment consists of two or more internal and external vendors.

![Cloud Architecture](image)

Services provided by the cloud can be categorized as infrastructure, platform, and software. These services are delivered to the devices over the network.

- **Software as a Service (SaaS)**: This service uses common resources and a single instance of application code as well as a database to support various customers simultaneously.

- **Platform as a Service (PaaS)**: It provides developers with a platform consisting of all the systems and environment with developing, testing, and deploying of applications as a service through the cloud.

- **Infrastructure as a Service (IaaS)**: Computer infrastructure as a service provides high flexibility and allows customers to use the latest technology.

### 2.2 Virtualization

Virtualization is a technology that permits a physical host to run multiple operating systems simultaneously. It adds a virtualization layer between the operating system and the physical hardware which are designed based on x86 architecture [26]. All systems are composed of the control unit, storage, arithmetic and logical unit (ALU), input and output devices. Virtualization layer handles multiple virtualized hosts by dynamically sharing memory, Device I/O and CPU which is made of ALU and control unit to access the hardware [13]. Virtualization simplifies the software development by enabling hardware abstraction and server consolidation which helps to replace single
physical hardware to several virtualized systems [26]. Thus, reducing cost, physical space, and energy.

Virtualization architecture can be either hosted architecture or hypervisor architecture. A hosted architecture runs virtualization layer as an application on top of the operating system whereas hypervisor architecture runs virtualization layer directly on the system. Hypervisor based architecture is more efficient than hosted as it does not reside on operating system and has direct access to the hardware, which improves performance [27].

2.2.1 Virtualization Techniques

The x86 architecture consists of four privilege levels as Ring 0, 1, 2, and 3 for handling access to the hardware. Operating system (OS) resides directly on the hardware, so they are at the level Ring 0 while the applications run on Ring 3. To virtualize the system, Virtualization layer as mentioned before must be placed between the hardware and operating system, which handles the shared resources [27]. They are three techniques to virtualize the system:

- **Full Virtualization**: Guest OS is completely abstracted from the hardware by the virtualization layer so it is unaware that it’s been virtualized. Full virtualization provides security for virtual machines and doesn’t require hardware or operating system help to virtualize privileged instructions. It uses binary translation which translates privileged instructions to user level instructions [28] [27]. It is a technique to run the unchanged OS that supports the platform being emulated [29].

- **Para Virtualization**: OS-assisted-virtualization or Para-virtualization can increase the performance of Full-virtualization by replacing instructions with distinct function calls known as hypercalls, which communicate with the virtualization layer. It is a technique to run several modified Operating System over a virtualization layer or Hypervisor.

- **Hardware-assisted Virtualization**: It automatically assigns the privileged calls directly to the hypervisor without the need of binary translation or paravirtualization. It is a technique which facilitates several isolated environments within a particular OS Kernel.

![Figure 3: System x86 Architecture](image-url)
2.2.2 Virtual Machine and Containers

Virtual machines are broadly used in cloud computing. Virtualization allows a physical device to have several virtual machines running concurrently on their operating systems. Each VM shares physical allocation of CPU, memory and I/O devices [26]. Virtualization uses a physical server and partitions it into virtual resources called Virtual machines [30]. Cloud platforms like Amazon provide their customers with virtual machines for running services. VM performance plays a significant role in cloud performance as all the workloads run in the VM [15]. Virtual Machines are managed by hypervisor which allows isolation for VMs is running on physical hosts and are also responsible for running different kernels in VMs [7].

A Container is an isolated virtual environment which includes a particular set of dependencies to run specific application [31]. A Linux container has a distinct approach than the hypervisor. It can be utilized as a substitute for a hypervisor-based virtualization [7]. Container based virtualization provides an alternative to virtual machines in the cloud. Containerization is a way where the user can run different isolated processors at a time. It uses only one kernel for multiple isolated operating systems [7]. Linux containers are light weight as they do not virtualize hardware but all the containers on physical host use a single host kernel efficiently using process isolation [7]. Unlike Virtual Machine which runs full OS, a container can contain a single process. A container which runs an entire operating system is known as a system container while the one who runs an application is called an application container. Application container does not have a separate IP address and consumes lesser RAM when compared to VM and system container [15].

![Figure 4: Virtual Machine vs. Container](image)

2.2.3 Hypervisor

Virtualization technology handles multiple instances of a virtual OS on a single system. This is made possible with the help of Hypervisor or Virtual Machine Monitor (VMM) which lies between the hardware and the operating system. The hypervisor is also known as virtualization layer that manages and hosts the Virtual Machines (VMs) [28]. It resides directly on the hardware, while the operating systems, applications, and VMs run on top of the hypervisor [32]. The guest is the virtual host which runs above the hypervisor, which has its OS and applications [26]. There are two types of Hypervisors:

- **Type 1 Hypervisor**, also referred to as bare metal or native, runs directly on the system hardware. They have direct access to the hardware resources and handles allocation of resources to guests. This hypervisor is used for server
consolidation to achieve optimized resource utilization. Xen, Hyper-V and VMware ESX are examples of type 1 hypervisors.

- **Type 2 Hypervisor**, also known as hosted hypervisor, runs on host OS as an application. It has less hardware issue as the OS is responsible for interacting with the hardware. They are also referred to as OS virtualization and are called containers in few scenarios. This type of hypervisor is useful when a single system requires the same set of OS activities to different users [22]. Virtual Box and VMware Workstation are examples of type-2 hypervisors.

![Figure 5: Hypervisor types](image)

2.2.4 Container-based Virtualization

Virtualization utilizing containers is a simpler alternative to hypervisor-based virtualization. A container is an isolated virtual environment which includes a particular set of dependencies to run a specific application. The process of isolation is implemented for containers at the level of OS for the host machine, thereby avoiding the overhead due to virtualized device drivers and hardware [31]. It is also known as operating system level virtualization as several isolated user-spaces run at the operating system level allowing abstractions directly to guest process [18]. OS level virtualization does not rely on hypervisor unlike Para and full virtualization. Modifications are made to the OS to isolate different instances within a single machine securely. The guest OS instances are often referred to as Virtual Private Servers (VPS) [33]. Containers using the same OS kernel result in a poor isolation in comparison to hypervisor-based virtualization [18]. Having a single instance of the kernel is advantageous as fewer resources are used because the OS has the overhead of multiple kernels [33]. OpenVZ and LXC are the trending container-based systems regarding performance that are used as an alternative to hypervisors [18].

2.2.4.1 OpenVZ

OpenVZ is for open-source virtualization and server automation developed by SWsoft [34]. OpenVZ creates several independent Virtual Private Servers (VPSs) on a particular physical server for sharing hardware. Each VPS operates and executes similar to the primary server for its applications having its root access, Users, IP addresses, memory, files, and applications. OpenVZ is suitable for servers with live applications due to its light overhead and efficient design. Unlike virtual machines, VPSs always run on the same OS kernel as the host system. OpenVZ is an operating system-level virtualization built of kernel namespaces, which allows each container to have an independent set of resources. It also
provides various network operation modes as Route-based, Bridge-based and Physical-based which lie at the network layer, data link layer and, physical layer respectively [18].

According to OpenVZ, a physical server is known as hardware nodes as they represent hardware units within the network. OpenVZ is based on Linux OS so it can run only on Linux distributions. Virtual private server (VPS) can be accessed by the container through /vz partition which is created during configuration. OpenVZ is installed in such a manner that user can boot the system either with or without OpenVZ support which can be chosen at the bootloader [35]. OpenVZ uses OS templates for creation of a container. An OS template is a set of packages from Linux distributions to populate one or more VPSs. Templates are usually created directly on the hardware node which consists of libraries, scripts, and programs required to boot and execute the VPS [35].

![OpenVZ Architecture](image)

**Figure 6: OpenVZ architecture ([35])**

OpenVZ consists of utilities for the command line to create, stop, start, destroy and migrate a container [35]. These utilities also help the user to manage resources and execute commands on containers. The `vzsplt` utility is used to create a configuration file which divides the available resources among the containers [6]. The `vzlist` utility can be used to list the existing VPS on the hardware node along with necessary information about the VPSs. The `vzctl` utility is the primary tool for VPS management which is used to perform operations on the container[35]. The `vzctl` internally makes use of `vzquota` to configure quotas. The `vzquota` utility is used to configure disk quota statistics for the VPSs and it also allows the user to configure per-user or group quotas inside the VPS [35].

WebVZ is a web-based management tool for OpenVZ which is an open source under GPL GNU license [34]. It has its web server and database engine. WebVZ allows the user to manage containers in an efficient way from the web browser instead of using command line tools. It helps to perform operations on a container and can manage resources, configuration files, OS-template and, user access [34].
2.2.4.2 LXC

LXC is a lightweight virtualization tool integrated into Linux kernel to run multiple virtual units. It allows Linux users to create and manage containers through its Application program interface (API) and tools. The primary intention of LXC is to create an environment similar to Linux installation without the need for separate kernel [36]. LXC is lightweight as it doesn’t virtualize hardware but the containers on the physical system use single host kernel efficiently through process isolation [7].

Linux containers are built on kernel namespaces feature for dealing high-performance computing clusters. Containers are isolated with the help of kernel namespaces and Control groups. Namespaces feature is used to create an isolated container that doesn’t have access to objects outside the container. It allows the processes in the container to expect that they are running on a regular Linux system. The root user in the container is not considered as root outside the container. LXC offers different types of network configurations, Route-based and Bridge-based. Resource management is implemented by using control groups (cgroups) [18]. Linux control groups are used to group processes, handle resource consumption and, limit the memory and CPU consumption of containers. Cgroups are used to resize containers by changing the container limits and it can also terminate all the processes in the container [15].

LXD is a container hypervisor which is built on top of LXC to improve the experience of the user. It uses Linux containers through liblxc to create and manage containers through command line tool. This tool allows the user to give an overview of all the existing containers on the network and also create new containers if necessary [37]. LXD manages containers through LXC’s API as it cannot virtualize anything individually. LXD considers full system containers and is not responsible for applications running inside the container.

2.3 Live Migration

Migration is a procedure of moving a virtual environment (VE) from one server to another, which is made possible due to the separation between workload and server hardware by the virtualization technology. Server load balancing, system maintenance, and energy saving are possible due to migration feature. To balance the load there is a need to migrate the virtual environment from one physical host to other [4]. Resources are efficiently utilized by transferring memory, storage, and network connectivity of virtual system from source to destination [38]. There are three approaches to performing migration of virtual systems [4]:

- **Cold Migration**: The guest OS is shut down, the virtual system is moved to another server and then the OS is restarted at the destination host.

- **Hot Migration**: The guest OS is suspended instead of shutting it down as in cold migration. The virtual environment is moved to another server and the guest OS is resumed at the destination. The applications state in the guest OS can be preserved during migration.

- **Live Migration**: Unlike hot and cold migration, memory pages of the VE are copied to the destination host while it is in running state. The downtime for the applications executing in live migration is reduced in the guest OS.
Live migration is a prominent feature of virtualization which facilitates dynamic resource management. It minimizes the downtime of VE when it is being migrated as the memory state is copied while the VE is running. Checkpoint and restart feature allows a running container to move from one server to another. The features allow to checkpoint the state of the container at a server and later restart it on the new server. This process involves transferring file systems of containers to another server. The state of the container is saved to a file and later copied to the new server. Now, the container is restarted at the destination from the file [10].

In live migration, Hypervisor must reflect the complete state of the VE at the destination server that involves processor, memory storage, and network. Memory migration is a crucial aspect during migration of VE where memory pages of the VE can be modified before the transfer is accomplished. They are two techniques to migrate memory state from source to destination host: Pre-copy and Post-copy memory migration.

- **Pre-copy migration**: In pre-copy phase, memory pages are iteratively copied from source to destination host when the virtual system is running at the source. At the first set, all the memory pages are copied while the subsequent iterations only the modified or dirty pages are copied. The modified pages are monitored by the hypervisor using a dirty bitmap. If the number of dirty pages reaches the limit, the pre-copy phase is terminated. In stop-and-copy phase, the VM is suspended at the source and the remaining dirty pages are transferred to the destination. Later, the VM is resumed on the target host [38]. The pre-copy memory migration procedure is shown in Figure 8.

![Figure 7: Pre-copy migration timeline ([39])](image)
• **Post-copy migration**: In the post-copy technique, the stop-and-copy phase is followed by pull phase for migration. The VM is suspended at the source system, and a minimal state of the VM is transferred to the destination. Later, VM is resumed on the target host where it fetches memory pages that are then transferred to the destination host [40]. A page fault occurs when the VM tries to access the memory pages which are not yet migrated. Maximum migration time and minimum downtime are achieved by this technique [13].

• **Hybrid migration**: It is referred as a particular case of post-copy migration where the post copy algorithm is preceded by a pre-copy stage. The most often used memory pages are transferred before the VM execution is resumed at the destination host. The performance degradation caused due to page faults in post-copy migration can be reduced by using this technique.

The performance of the migration can be measured by using metrics, Downtime and total migration time. Downtime is the period during which the services provided by the migrating VM are unavailable or not responding to requests. Total Migration time is the duration when the migration process has started to the time when VM is made available to the destination server. These metrics show the impact of migration on the system as the resources used during migration is not available to perform other tasks [38].
2.3.1 OpenVZ live migration

Container-based virtualization is an ability to run multiple isolated sets of processes, known as containers, under a single kernel instance. Due to this isolation the complete state of the container can be saved and later restarted. Checkpoint and restart allow checkpointing the state of running container and then restart it at the same or different container. This feature moves a running container from one server to another that is known as live migration [10]. Containers continue to run during the live migration process, which results in shorter container Downtime and longer Migration Time [41].

The process of live migration in OpenVZ consists of several steps: the first synchronization, suspend, dump, second synchronization, copy dump file, recover, resume, and stop and clean [10] [42].

- **First synchronization**: Container file system is moved to destination server with the help of rsync utility
- **Suspend + dump**: Freeze all the resources and disable networking. Later, dump the resources to a file on the disk.
- **Second synchronization**: During the first synchronization the container is still running, so a second synchronization is performed after freezing the resources to update the memory pages in the container.
- **Copy dump file**: The dump file is transferred to the destination host
- **Recover**: Create a container from the dump file on the target host with the same state as in the source host
- **Resume**: Resume the containers execution on the destination host
- **Stop and clean**: Destroy the container on the source host and remove all the files related to the container.

2.3.2 LXC live migration

LXD is a container hypervisor that is built over LXC for creating and managing containers. LXD has an advantage where it automatically creates a bridged network, lxcbr0 for assigning a network to the containers. LXD is based on images so containers are created from an image to use the facilities of LXD [43].

Migration of containers is made possible through checkpoint/restore feature which checkpoints the state of the container and later restored. Migration can be performed by checkpointing the state of container on the host system and then restoring it on destination host [44]. Checkpoint and restore functionality is implemented by CRIU (Checkpoint/Restore In Userspace) in Linux [45]. P.Haul checks and orchestrate all the checkpoint/restore steps. It provides the engine for containers live migration using CRIU and organizes memory pre-copy or post-copy. LXC live migration involves:

- **Freeze + save state**: Freeze the container at the source node blocking memory, process, files system and network connections and save the state of all the objects in the container.
- **Copy state**: Copy the state to the destination node.
- **Restore + unfreeze and resume**: Restore the apps from the images.
- **Cleanup**: If migration is successful, kill the stopped tasks on the host.

![Figure 11: Live migration process](image)
3 RELATED WORK

Due to high growth in cloud computing [25]. Various researches have been done in this field. In [12] the authors proposed architecture to use market-oriented allocation of resources within clouds. The cloud has limited support for market-oriented resource management and the interaction protocols need to be extended to support interoperability between different cloud service providers. In [3] the authors made a taxonomy of cloud computing systems and later conducted a survey of the existing cloud services developed by Google Amazon etc. The taxonomy provides the researcher with an idea of existing cloud systems and challenges. The survey results were used to identify similarities and differences in various cloud computing approaches. They also found out areas that require further research. The author further mentioned about the evaluation and improvement of the existing and new cloud systems. In [1] various obstacles that affect the growth of cloud computing are listed and they also stated the possible techniques to overcome the obstacles. In [2] the authors made some efforts to improve the level of security in cloud computing by surveying the existing security models to make customers realize the importance of safety in the cloud.

The usage of containers has increased in the recent times as they have been the solution for cloud delivery problems [46]. In [15] the authors have explored the performance of the virtual machine and containers based on storage, networking, and memory of the CPU. Their results show that containers have similar or better performance than VMs in different scenarios. In [16] efforts are made to compare container and VMs based on high availability. Results show that containers have better high availability (HA) features than hypervisors based systems. In [7] comparison is based on performance and scalability. Results show that containers outperformed VMs due to its better utilization of resources. Containers can be used for application distribution to diminish resource overhead while VMs can be utilized for running application with critical business data.

Live migration performance is measured with downtime and total migration time. A balance between these two parameters is an ideal solution. Various researchers have made efforts to enhance the performance of live migration. In [14] the authors proposed an approach which uses logging and replay for live migration of Docker containers. This technique has reduced the Downtime and Total Migration Time to a certain extent under different scenarios when compared to traditional VMs. This approach consists of three stages copying the image file, replay and iterative log, and stop and resume. The images file of the source container is copied to the target host. The base image of the container is shared and the additional layer is duplicated that results in less amount of data to be transferred. Later, log files are copied iteratively until the size of the file reaches the threshold value. Now, source container is stopped and the container is resumed at the target host.

The parallel application will abort whenever a computational node fails. In [42] authors made efforts to decrease the fault rate of the application by migrating a process from a node which is failing to a node which is healthy. The experiment is done on OpenVZ platform with parallel applications with multiple processors running on it. They have succeeded to migrate containers without severely affecting the correctness of the parallel application. Live migration can be used proactively for fault tolerance as a viable strategy to increase the mean time to failure of an application. High availability is often more important than performance for cloud computing. Live migration enhances the availability but it has an overhead. In [47] a lightweight live
A migration approach has been proposed to coordinate system migration and input replay efforts that aims to reduce the overhead while providing comparable availability.

Virtualization is a technique which divided computer resources to several independent environments [29]. Kirill Kolyshkin has stated about the various types of virtualization: emulation, para-virtualization, and OS-level virtualization along with its advantage and disadvantages. In [17] the authors measured and analyzed the performance of OpenVZ, Xen, and KVM which represent OS-level virtualization, paravirtualization, and full virtualization respectively. The results show that OpenVZ has better performance than Xen followed by KVM which also states that OS-level virtualization and para-virtualization have an advantage over data intensive applications. In [33], Virtualization is commonly used for efficient utilization of computing resources within servers. Different virtualization types are compared between OpenVZ, Xen and VMWare Server for high-performance computing applications. The results show that OS-level virtualization implemented by OpenVZ has the best performance when compared to other techniques mainly for MPI scalability.

Container-based platforms are taken into consideration where performance is a critical factor. Virtualization for resource-intensive applications provides an improved level of manageability, scalability, security and also utilization of resources. Porto Alegre had made an effort to compare container-based platforms, LXC and OpenVZ, considered when running on MapReduce clusters concerning performance and manageability [18]. Results proved that LXC has a better tradeoff between performance and manageability. In [48], container-based systems, OpenVZ and LXC, are measured and compared based on network virtualization. The results are a comparative analysis of the convergence of routing protocol and provide information about utilization of these tools on a routing as a service platform.

The performance of VMM plays a significant role in improving the design and implementation. In [19] virtual machine monitors, OpenVZ, Xen, and KVM, which are based on different virtualization techniques are evaluated. Qualitative and quantitative comparison of these three VMMs is provided. Their performance is measured as a black box and the results are analyzed as a white box. Isolation techniques in any platform which restrict resource have an impact on the performance of virtual machines [20]. This literature deals with network throughput impact on hardware virtualization approaches, OpenVZ, KVM, Xen, Virtual Box, and VMware on shared host and bare system. The impact is compared on shared host and bare host. The results explained the effect of virtualization while considering the throughput on a single network interface.

Sogand.et.al.[28] has measured the performance of hypervisors, KVM, VMware and XenServer, based on response time, CPU and disk utilization of an application in real-time. The performance of live migration for these hypervisors is measured regarding Downtime and total migration time. Results show that Xen has more CPU utilization and less downtime during live migration when compared to other hypervisors. So, a single hypervisor does not have the best performance in all the scenarios. In [21], Performance is compared between the hypervisors; XEN-PV, XEN-HVM, and OpenVZ based on downtime, total migration time, CPU and disk utilization. Changing the packet size also compares performance. Results show that OpenVZ has higher CPU utilization and lower total migration time and downtime than remaining chosen hypervisors for this experiment.
4 Methodology

Research methods are conducted to answer research questions [49]. The chosen research method depends on the type of problem we are addressing; an appropriate method must be selected to get the best answer. For developing objectives and selecting methods, validity and reliability should be taken into consideration. The relationship between what you want to measure and what is required to measure is known as validity. The accuracy of measuring the method is known as reliability [49]. The results obtained from the research should be valid, so research method is used to ensure the validity of data.

The research method followed for conducting this study to solve the research questions are classified as three stages.

1. Literature review for getting familiar with the topic and gain extensive knowledge in the present field of the interest.
2. Construct a test environment where the experiment is performed.
3. Analyze the results observed in the experiment.

The experiment is chosen as a research method. It focuses on investigating few variables and the ways in which these are affected by the experiment conditions. Experiment are often done by implementing a model of some system and running simulations to see how the model is affected by different variable [49]. Total migration time and Downtime are used as metrics to measure the performance of live migration in OpenVZ and LXC which were selected by conducting a literature review. As the factors which affect the performance are well known in advance before carrying out the experiment, so this method serves as a better way to answer the research question. The process is repeated several times to get valid data for measuring the performance efficiently. Experimentation is performed to help us better evaluate, predict, understand, control, and improve the software development process and product [50].

Exclusion criteria:

- Survey is chosen when the relatively well-known phenomenon is considered. This method is not preferred as it only gives the opinion of the practitioners. This data wouldn’t be sufficient to compare the performance of this software. Different metrics affect the performance of the live migration, which cannot be evaluated by using this method.

- A case study is preferred as a research method for exploratory studies which are indefinite in nature. Since the objective of the present research is to evaluate the performance of container engines, this method does not meet the requirements.

- Action research is used to solve a real-world problem. It is implemented when a result of an action is to be known. This method can’t be used as the objectives of the research is to evaluate the performance of live migration.
4.1 Literature Review:

Literature review refers to identifying and outlining the studies about a particular field in the research area. These research studies are comprised of articles that facilitate the researcher with a framework for analyzing the topics. The primary objective of literature can be understood as a search for finding numerous research articles for a better understanding of the current research study [51]. Initially, a literature review has been conducted to obtain a gap in the present research area and then establish an association between the findings of this study with the existing literature. Related work as described briefly in Chapter 3 is formulated by conducting a literature review in a systematic way to collect, evaluate and summarize the literature. The steps involved in this process are [52]:

- **Step 1**: Identify keywords to collect literature required to understand the present research topic. Keywords may emerge in the identification of a research topic or while preliminary reading. The primary keywords in this research are “live migration” and “virtualization”.

- **Step 2**: Appropriate keywords are chosen by considering the meanings, abbreviations and alternative spellings for the identified keywords. Sophistically valid and understandable search strings are constructed using Boolean ANDs and ORs [51]. Search strategy has been organized in computerized databases for obtaining the study related to the research area. The literature study often involves articles, journals, books, etc. The search strategies are usually iterative with numerous combinations of the search words derived from the research questions that were proposed [51]. Scientific databases, like Inspec and Google Scholar, are used to gather extensive literature in the present research area. Inspec was selected, as result of the search string can be shortlisted using controlled vocabulary. Google scholar has been chosen as its search includes results from literature across many sources and disciplines [52].

- **Step 3**: Relevant articles are selected from the obtained results, which formed the foundation for the literature review. More keywords were analyzed from these selected articles to get deeper into the research area. These keywords were “containers”, and “Virtual machines”.

- **Step 4**: More articles were searched in the databases with the newly deducted keywords. Articles which are closely related to the research area are selected. This selection is based on a quick review of the abstract and the results of the retrieved articles.

- **Step 5**: Filter the initial group of articles or research papers by using inclusion and exclusion criteria to those that are central to the present research area. After analyzing all the selected articles, only the necessary articles are retained, which will make a useful contribution to understanding the existing research area.

- **Step 6**: These articles are tabulated along with its contributions to various topics in the research area. Each research paper was given a rating to the corresponding subtopics of the present research area according to the depth of information of certain sub-topic as shown in Figure 36 in Appendix A. This information was helpful while documenting background and related work in this research area to easily navigate between various papers which correspond to the same topic.
The literature review was further made strong by implementing the backward and forward snowballing method and carefully observing the articles that were selected [51]

4.2 Experiment

4.2.1 Test Environment

Live migration works only between servers with identical CPU architecture. The testbed consists of three servers with identical configurations, which are used to conduct the experiment. Each server configurations are mentioned in Table 1. The servers are connected to the same subnet. Two servers are used for performing live migration of containers and the third server is used for generating stress on the container through Cassandra stress tool.

<table>
<thead>
<tr>
<th></th>
<th>Manufacturer</th>
<th>Dell Inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Operating system</td>
<td>GNU/Linux</td>
</tr>
<tr>
<td>3</td>
<td>Distributor</td>
<td>Ubuntu 16.04.1 LTS</td>
</tr>
<tr>
<td>4</td>
<td>Architecture</td>
<td>X86_64</td>
</tr>
<tr>
<td>5</td>
<td>Codename</td>
<td>Xenial</td>
</tr>
<tr>
<td>6</td>
<td>Kernel release</td>
<td>4.4.0-36-generic</td>
</tr>
<tr>
<td>7</td>
<td>Virtualization</td>
<td>VT-x</td>
</tr>
<tr>
<td>8</td>
<td>CPU(s)</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>Model name</td>
<td>Intel(R) Xeon(R)CPU E5-2420 v2 @ 2.20GHz</td>
</tr>
<tr>
<td>10</td>
<td>Cache size</td>
<td>15360 KB</td>
</tr>
<tr>
<td>11</td>
<td>Memory</td>
<td>24637832 kB</td>
</tr>
<tr>
<td>12</td>
<td>Disk size</td>
<td>279.4Gb</td>
</tr>
<tr>
<td>13</td>
<td>CPU MHz</td>
<td>1205.789</td>
</tr>
</tbody>
</table>

Table 1: Server Configuration

4.2.2 Initial Setup

The Hypervisor was installed in two servers which were allotted for performing a live migration. The hypervisor acts as a virtualization layer between the operating system and hardware thus allowing it to create containers with an isolated environment. OpenVZ and LXC hypervisors (Type 2) are used for performance comparison during live migration of containers. LXC is available directly in Linux distribution with a minimum kernel requirement of 3.12 [36]. The server is built on Ubuntu, a Linux distribution, with 4.4 kernel, so LXC package can be installed by using the command: `apt install lxc`.

OpenVZ can be installed directly through Linux repositories only on Ubuntu 8.04 (Hardy) [53]. For the later versions of Ubuntu, there is no direct support for OpenVZ. However, they are continuous developments for OpenVZ in RHEL and CentOS distributions [53]. In order to install OpenVZ kernel on Ubuntu, its repository and RPM packages are installed from the OpenVZ official website[34] along with GPG key to sign RPM packages. Now, OpenVZ kernel and its tools, vzctl, vzquota, and ploop packages, are converted to Debian files and later installed on the system. Once the packages are installed, the system is rebooted into OpenVZ kernel. Cassandra stress tool (version 3.0.4) was installed in the third server which is used to increase the load of the containers by using its public address.
Containers were installed with Cassandra (Version 3.0.4), which is similar to the version installed on the third server. Containers assigned with a private address can be accessed only through the host server. So, Yet Another Markup Language (YAML) file must be edited with the IP address of the container and the host server. Cassandra-stress tool on the third server is only connected to the container through the host server as they are binding with the private address of the server. Now Cassandra stress tool can generate load on the containers. But, the problem with the private address is that during live migration, the connection will be shifted to the destination server. This change in the server for the container loses its connection with the Cassandra-stress tool, which stops the load generation in the container at the target server. Public addresses, which are not bound to any servers like a private address, are used in the experiment to measure the impact of migrated container on the destination server. Now, the Cassandra stress tool can continuously generate load on container even after migration.

![Live Migration of Container](image)

**Figure 12: Testbed**

### 4.2.3 Migration

#### 4.2.3.1 LXC

Live migration works in LXC only between servers with identical CPU architecture. For performing live migration of Linux Containers, it requires both the servers to have Linux kernel higher than 4.4, CRIU 2.0 and LXD running directly on the host [44]. The steps followed in the experiment for live migration of LXC containers are based on the websites of the official developers of CRIU and LXC [44]. By considering these requirements, the experiment is conducted on the similar server which is operated with Ubuntu 16.04 (4.04 kernel) as shown in Table 1. LXD is a container hypervisor which is built on top of LXC for creating and managing containers. LXD and CRIU are also included in the Ubuntu 16.04 repositories which are directly installed.

The initial configuration of LXD is performed using the command: `lxd init`, which prompts to select a backend storage device and network settings for the container. LXD has an advantage of automatically creating a bridged network for assigning a
Containers are set up with Ubuntu template and later started using the lxc launch command. Here, C1 is the container name as shown in Figure 13.

- **lxc launch C1 -t ubuntu**

Now a profile is created for setting container configurations to support migration which is shown in Appendix B. This profile includes security tweaking and enabling CRIU support to the container which is to be migrated to another host. LXD, by default, listens on Unix socket in both host and destination servers. For live migration to work, both the host and target servers are configured to listen on TCP port 8443, which allows the servers to communicate with each other through TCP connection. Later LXD servers are assigned with passwords to ensure security.

- **lxc config set core.https_address [::] 8443**
- **lxc config set core.trust_password password**

The connectivity between the servers was set using the remote command to store the IP-address of host and destination servers to a particular name. Here, lxd1 and lxd2 are the names given to the source and destination host based on their IP-address as shown in Figure 13.

- **lxc remote add lxd1 ipaddress:8443**
- **lxc remote add lxd2 ipaddress:8443**

Now, using these connection settings, destination server must copy the profile with container configuration to ensure connectivity to the container after migration. The container can now be migrated using move command.

- **lxc profile copy C1 lxd2**
- **lxc move lxd1:C1 lxd2:C1**

The procedure involved to install and perform live migration [44] are mentioned clearly in Appendix B.
4.2.3.2 OpenVZ

The system must be booted into OpenVZ kernel to use its features, which acts as a hypervisor for managing containers. After booting into OpenVZ kernel, the templates necessary for the creation of containers are downloaded from an official directory of OpenVZ [34]. The container is created with the container-id and OS template by using the `vzctl` command. Here 101 is the container-id and ubuntu is the template assigned to the container.

- `vzctl create 101 –ostemplate ubuntu`

Unlike LXC that configures networking automatically for Containers, OpenVZ uses `vzctl` command for assigning IP-address and name servers to the containers. To enable internet access to the containers IP masquerading must be configured on the server.

- `vzctl set 101 --nameserver " " --save`
- `vzctl set 101 --ipadd " " --save`

Live migration of OpenVZ containers can be performed only when root user of host server can log into destination server through SSH without being asked for a password. So, a public key is generated on the host and is appended to the authorized keys of the target server. The OpenSSH Daemon (`sshd`) configuration file must be modified to permit root login. Now, live migration can be performed by using OpenVZ migrate command. The steps involved for installing and performing live migration in OpenVZ [34] is explained in detail in Appendix B.

- `vzmigrate --online destination_ip container_id`

4.2.4 Factors

The system parameters can be divided into two categories: Constant and Varied, during the evaluation. The parameters which will vary during the evaluation are called factors and its values are known as levels [40]. Instances run under various workloads as they depend on the service and the application in a cloud environment [40].

The performance is measured for different CPU loads of the container to analyze load impact on the migration process. Live migration is performed to balance the load on the servers by migrating containers from overloaded to regular load servers. The CPU load of the container plays a vital role in the migration process. So, the experiment was performed on different loads for better analysis. The load is generated in the containers using Cassandra-stress tool on the third server.

The load generated by the Cassandra-stress tool can be varied using thread count. Initially, stress command is run for few minutes to find the threshold value for a thread which is considered as 100% load. In our case thread count, 455 is the threshold value for the write operation. Now, to get 100% load on the container, the stress tool in the third server runs the write operations with the thread count as 455. Similarly, the experiment is performed for the thread count 110 to measure the performance at different load.
4.2.5 Performance Metrics

The performance of the hypervisors while live migrating the containers are evaluated using metrics. Downtime and Total Migration time are the primary metrics which are used to compare the performance.

Migration time[11] is the time required to transfer the dirty pages from the source to the destination, which is calculated using time parameter for the migration commands. The migration commands of OpenVZ and LXC start the migration process and the control to the terminal are returned only when the command has finished its execution, which is the completion of the migration process. So, measuring the time taken for the execution of these commands using time parameter would give the total migration time.

- \textit{time lxc move lxd1:C1 lxd2:C1}
- \textit{time vz migrate -online destination_ip container_id}

Total migration time is also calculated by using ping utility. When migration process has started, simultaneously ping is performed to the destination host from the source server with a time interval of 0.5s. The number of packets resulted during the execution of migration gives the total migration time. The results from both the methods are used to interpret the data efficiently instead of using the single method and a single set of data.

Downtime is the time between the suspension of migrated VM at the source and its resumption at the destination node [11]. Ping utility is used to measure downtime of the container. When the migration has started on the source server, ping is performed with 0.01 interval from destination server to the container IP-address. After completion of migration, the packets received by the destination node are analyzed to find the number of packet loss which gives the downtime of the container i.e. the number of ICMP requests where no response is obtained [40].

Along with these metrics, CPU, Disk Utilization, and load average are also considered as they affect the performance of the server. These metrics are measured before, during and after migration of containers to analyze the performance difference between the hypervisors efficiently.

To measure these metrics \textit{sar} and \textit{iostat} utilities are used. A part of \textit{sysstat} package is the \textit{sar} command which is capable of observing and monitoring the actions for a long time [28]. The output generated by the \textit{sar} command is sent to a CSV file and later the data is interpreted in Excel to plot the result graphs. \textit{iostat} command monitors the input and output device by observing the time device and averages their transfer rates [28]. CPU utilization and load average are measured using \textit{sar} command while Disk utilization is measured using \textit{iostat} command which gives device utilization report per physical device. Microsoft Excel software is used for illustrating the graphs by plotting the data stored in the CSV files.
5 RESULTS

The results of the literature review are stated in Background and Related work chapters. Background section consists of basic concepts for understanding the concept of live migration and further the procedure for live migration of containers in OpenVZ and LXC are also explained briefly. Related work consists of the studies related to the present research along with the author contributions. The results for the experiment are described in the following sections.

5.1 Case 1:

OpenVZ is installed on two servers to support live migration of containers and Cassandra NoSQL database is installed on the third server, which was used to measure the performance of live migration of containers with different loads. Cassandra-stress tool was used to increase the load on the container to be migrated. The maximum thread count was chosen as 450 since from 450 to 600 threads, the write operations were found constant with maximum write operations.

Cassandra NoSQL database is installed in OpenVZ container to facilitate the Cassandra-stress tool for increasing the load on the container through write operations. The number of threads chosen for measuring the performance of live migration is 150 and 450. The command used for stressing the load on the container is as follows

```
./cassandra-stress write duration=15m cl=ONE -pop dist=UNIFORM(1..50000000) -rate threads=450 -node ip_address;
```

Containers are made running with Cassandra NoSQL database for seven minutes before performing live migration followed by seven minutes after migration at the destination node. CPU utilization, Disk utilization, and Load average are measured using `sar` and `iostat` commands. When the experiment was conducted, all the remaining processes were killed before evaluating the performance to exclusively measure the impact of migration on the server. The results for these metrics are illustrated in graphs and stated below.

5.1.1 Scenario 1:

The number of threads was initialized as 150 in the Cassandra-stress tool command. Live migration was performed at 360 seconds which can be seen in Figure 14. The experiment was repeated five times to get the accurate results and the mean of Total Migration Time is 80 seconds with a standard deviation of 1.86. The mean value of Downtime is 14.7 ms with a standard deviation of 0.785. A special parameter ‘-v’ was used in migration command to get the time for different phases of migration.

```
vzmigrate -v --online destination_ip container_id
```

The average values for different phases of migration, suspend and dump, pre-copy after suspend, copy dump file and, undump followed by resume are 5.62, 1.47, 4.22, 2.97 respectively. The total of these phases gives the suspended time which is 14 ms. This value is also considered in the dataset of Downtime for all the iterations to get accurate results.
Due to stress on the container, the maximum CPU utilization was recorded as 47% at source and 36% at the destination server. Average CPU utilization before live migration was 8%. During live migration, it reduced to 0.8% and after completion of migration process it got further reduced to 0.2%. At the destination server, average CPU utilization was 0.5% before migration, which gradually increased to 2.43% during the migration process, and further, it has moderately increased to 7.6% after the migration process.

![CPU Utilization Graph](image)

**Figure 14: CPU Utilization for OpenVZ (threads = 150)**

The maximum Load average for the past one minute was recorded as 8.86 at the source and 10.85 at the destination as shown in Figure 15. The mean value of Load average for one minute at the server of origin is 2.76 before migration which increased by 5% to 3.25 during migration and later reduced by 10% to 0.28 after the container has been migrated to the destination server. At the target server, the Load average for past one minute is 0.5 and during live migration it gradually increased to 0.8 and after the migration of running container, CPU load increased by 6% to 5.5. The average load for past one minute, five minutes and fifteen minutes for the source and destination are shown in Appendix A.

![Load Average Graph](image)

**Figure 15: Load average for OpenVZ (threads = 150)**

The maximum Disk utilization was recorded as 334888 writes/second at the source and 301192 writes/second at the destination server. At the source where the container is
being stressed by Cassandra-stress tool, the average Disk utilization was 6224 writes/sec which remarkably increased by 50% to 12375 writes/sec during live migration and later reduced to 276 writes/sec after migration. At the destination server, the average disk utilization was 243 writes/sec before migration which drastically increased to 109951 writes/sec during live migration and later reduced to 6126 writes/sec after the migration.

![Disk Utilization](image)

**Figure 16: Disk Utilization for OpenVZ (threads = 150)**

### 5.1.2 Scenario 2:

The number of threads used for stressing the load on OpenVZ container was 450. Live migration was performed after seven minutes at 420 seconds. The mean value calculated for different iterations for Total Migration Time is 74 seconds with a standard deviation of 1.7. The mean value for Downtime is approximately 14 milliseconds with a standard deviation of 1.03. The average values for different phases of OpenVZ migration process obtained through a special parameter are 6.61s for suspending and dump phase, 1.54s for Pre-copy after suspend, 4.40s for coping dump file and finally 2.45s for un-dumping and resuming. The total gives the suspension time which is 15s.

The maximum value for CPU utilization was recorded as 30% at Source server where the container is loaded through stress tool from the third server and 29% on the destination server when the container has been migrated as shown in Figure 17. The average CPU utilization was 4.2% before migration, which increased to 5.1% during migration and gradually decreased to 0.25% after the process as the overloaded container is being migrated. The source server would get back to its normal state. At the destination server, the average CPU utilization before live migration is 0.22% and during the migration process, it has been increased to 2.3% and further increased to 3.42% after the migration as the loaded container is migrated to this server.
The Disk utilization has increased to a great extent during the process of live migration as shown in Figure 18. The maximum Disk utilization at the server where the container is being stressed is 267048 writes/sec while it is 293888 writes/sec when the container has migrated to the destination server. The average Disk utilization of the source server before live migration process is 3241 writes/sec, during the migration process it increased to 15050 writes/sec and finally after the migration is 30 writes/sec. The average disk utilization of the destination server before, during and after migration process are 22 writes/sec, 107026 writes/sec and 2505 writes/sec.

The maximum Load average for the past one minute is almost similar to the source and destination server which is around 5. The average load of source server for the past one minute is 1.7 before live migration process and during migration it increases to 2.3 and further it decrease to 1.2 after the migration process. At the destination server, initially the average load is 0.46 and during the live migration process, it increased to 1 and further it increased to 2 after the migration process.
5.2 Case 2

LXC is installed on both the servers allocated for hypervisors and Cassandra-stress tool is installed on the third server. All the other processes are killed before performing live migration to maintain consistency of data and make sure that the effect on Disk utilization, CPU utilization are due to migration process exclusively. Cassandra is also installed in the LXC containers which were set to be migrated to permit Cassandra stress tool to increase the load of the container.

5.2.1 Scenario 3:

The number of threads used by the Cassandra-stress tool to increase the load is set to 150. The mean value of Total Migration Time is 78 seconds with a standard deviation of 1.7. The mean value of Downtime is 8 milliseconds with a standard deviation of 0.8.

The maximum CPU utilization for to 8% and finally after total migration it further increased to 58% at the source server where the container is loaded by the stress tool is 85% while for destination server is 78% when the container is migrated to this server. At the source server, the average CPU utilization before live migration is 60% as the container is loaded and during the migration of container, it reduced drastically to 7% and after the container is migrated it further decreased to 0.9%. At the destination server, before the container is migrated to this server average CPU utilization is 0.1% and while the container is migrating the value increased.
The maximum Load average for past one minute was recorded as 32 on the source server and 37 at the destination server when the loaded container resides. The Load average for the server of origin before migrating the container is 20 and it reduced to 4 during the migration and further decreased to 0.9 after the container is moved. At the destination server, the Load average for the past one minute is 0.03 before the container is migrated to this server, and during the migration, the value increased to 1.6 and further increased to 24 after the container has successfully migrated to this server.

![Figure 21: Load average for LXC (threads=150)](image)

### 5.2.2 Scenario 4:

The number of threads is set to 450 for the Cassandra-stress tool to increase the load of the container. The live migration process has started at 290seconds. The mean value for Total Migration Time is 72seconds with a standard deviation of 1.8. The mean value for Downtime is 7ms with a standard deviation of 0.8.

The maximum CPU utilization for the source server was recorded as 86% and 84% for the destination server as shown in Figure 22. The average CPU utilization of source server before, during and after the live migration is 16%, 11% and 0.3% while for the destination server is 0.2%, 2%, and 32% respectively.

![Figure 22: CPU utilization for LXC (threads=450)](image)

The maximum Load average for the past one minute is 42 for the source server and 50 for the destination server. The Load average for the source server is 16 before
migrating the container, which decreased to 11 during the process of migration, and further got reduced to 0.3 after the migration. The average load for the destination server before migration is 0.26, which has increased to 2.2 during the migration of containers, and finally increased to 32 after the migration.

Figure 23: Load average for LXC (threads=450)

The maximum Disk utilization for the source was recorded as 267456writes/sec while for the destination server is 324608writes/sec as shown in the figure. The average Disk utilization of the source before live migration was 59290writes/sec which decreased to 62writes/sec during the process of migration and finally reduced to 24writes per second after migrating the container. The average disk utilization of the destination was 28930writes/sec before live migration, 227976writes/sec during live migration and 97801writes/sec after the migration process.

Figure 24: Disk utilization for LXC (threads=450)
6 ANALYSIS

Downtime:

From the results, it can be analyzed that LXC has lower downtime when compared to OpenVZ. The mean values are used to illustrate the graphs as shown in Figure 25, which conveys the significant difference between the hypervisors.

![Downtime Chart](image)

**Figure 25: Downtime for LXC and OpenVZ**

A Box plot is a five-number summary, which can be used to display the center and variation of the outcome values in the dataset through graphs [54]. The five-number summary includes a minimum value, maximum value, and quartiles in the given data set. The box plot graphs are illustrated for values obtained by taking the iterations of the data for the both Downtime and Total Migration Time.

The Box plot graph of the Downtime metric for thread count 150 and 450 are shown in Figures 26 and 27, where the significant difference between the Hypervisors (OpenVZ and LXC) is clearly visible. If the boxes in the boxplot overlap, then the difference is measured by calculating the overall visible spread and the distance between the medians. But, the boxes are well separated from each other showing a huge difference between the hypervisors. From the graphs, it can be concluded that LXC has less downtime when compared to OpenVZ.
Figure 26: Box plot for Downtime (Threads=150)

Figure 27: Box plot for Downtime (Threads=450)

**Total Migration Time:**

The total migration time for both the hypervisors is almost similar but they differ in a small scale. The mean values of the migration time in different scenarios is used to illustrate the graphs as shown in Figure 28 which represents the significant difference between two Hypervisors.

Figure 28: Total Migration Time for OpenVZ and LXC
A box plot is also used to find the difference between the Hypervisors in an efficient way through graphs by considering the minimum value, maximum, and quartiles in the observed dataset. The median is the vertical line bisecting the box. The medians of the boxes representing LXC and OpenVZ are not overlapping, which represents that there is a significant difference between the Hypervisors. OpenVZ has higher values for Total migration time in the Figures 29 and 30, which show that LXC has lower Total Migration Time when compared to OpenVZ.

![Figure 29: Boxplot for Total Migration Time (Thread=150)](image1)

![Figure 30: Boxplot for Total Migration Time (Thread=450)](image2)

**CPU Utilization:**

LXC has higher CPU utilization, so higher CPU load when compared to OpenVZ, as mentioned in scenario 1 & 2. OpenVZ has a maximum utilization of 47% and 37% and maximum load average for one minute around 8, while LXC has a maximum of around 85% utilization and 32 Load average for past one minute.
CPU utilization of OpenVZ decreases gradually at the source and increases gradually at destination during and after live migration while for LXC, during live migration, CPU utilization decreases drastically at the source and increases drastically at the destination.

Figure 31: CPU Utilization for LXC and OpenVZ

Figure 32: CPU utilization for LXC and OpenVZ at Destination

Figure 33: CPU utilization for LXC and OpenVZ at source
Load Average:

The Load average of OpenVZ at source gradually increases during live migration and decreases after the container has migrated, while for the destination, it gradually increases. A load average of LXC is similar to CPU utilization which gradually increases at target and decreases at the source.

![Load - Source](image)

Figure 34: Load average at source for LXC and OpenVZ

![Load - Destination](image)

Figure 35: Load average at destination for LXC and OpenVZ

Disk Utilization:

Disk Utilization on the origin server of both OpenVZ and LXC is higher than the destination server due to the write operations generated by the Cassandra-stress tool. At the time of live migration, the disk utilization increases gradually at the source and drastically on destination. After the migration of container, disk utilization (writes/sec) drastically decreases at the source and gradually decreases at the destination. As the write operations are generated in the container, disk utilization shifts to the destination along with the container.
LXC has an advantage over OpenVZ in total migration time and downtime for containers with lesser data. There is only a slight decrease in migration time but it has less downtime when compared to OpenVZ. However, LXC failed to migrate containers with a higher load which resulted in repeated errors while OpenVZ can easily migrate a container with any load. The major advantage of OpenVZ is it displays the users with the process of migration and also represents time taken for each phase. LXC live migration is a black box where the user has to wait for the control to return to the terminal which represents the successful termination of the migration process.
7 DISCUSSIONS AND VALIDITY THREATS

This section discusses the findings of the present research study with the existing literature. The present study aims at comparing the performance of OpenVZ and LXC hypervisors during the process of live migration. The Experiment is conducted on a testbed by migrating a container with different loads in OpenVZ and LXC respectively. The container load is varied similar to a real world scenario by using Cassandra NoSQL database. The metrics used for measuring the live migration are downtime and total migration time. Further, CPU utilization, Disk utilization, and Load average are also measured during the process of migration to analyze the performance of hypervisors.

S. Shirinbab et al. have compared the performance between KVM, VMWare and, XenServer based on the metrics CPU Utilization, Disk Utilization (number of write operations per second), downtime and total migration time [28]. The methods used to evaluate these metrics during live migration has been considered in this study to evaluate the performance of OpenVZ and LXC during live migration. In [28], the author stated that no single hypervisor has the best performance when considering all the parameters. The existing literature results show that Xen has higher CPU utilization and lower downtime during migration when compared to other hypervisors considered in the experiment. Similarly, the results of the experiment performed in the present study proved that LXC has higher CPU utilization and lower downtime.

Factors that affect the performance of each phase of live migration are virtual machine size and CPU load. The experiment in [40] was done in cloud environment on different CPU loads of 25%, 50%, 75%, and 100% which is obtained by using the stress-ng tool. This tool directly forks the CPUs to a certain constant load assigned by the user. To get a 50% load on a server with 12 cores, the user assigns CPU load as six which makes the stress-ng tool stress 100% on 6 cores which represent overall 50% load on the server. The CPU load is constant to the assigned value until the stress-ng command is terminated, but this is not realistic in a real world where the load changes dynamically from time to time. In the present study, Cassandra-stress tool was used to generate read/write operations in the container to increase its load similar to a real world scenario. The stress tool was run for few hours and from 450 to 600 threads the write operations were constant. So as an average, the thread value was set to 450 where the server got the highest write operations. So, the high load was considered as 450 threads and thread count 150 was also considered to measure for different workloads on the server.

The authors in [48] concluded that LXC with bridge based connectivity is faster than OpenVZ with an average of 20% time lower than OpenVZ to perform a task. Connectivity for the containers is found easy or straightforward in LXC where it configures a bridge with the default settings for network access. OpenVZ doesn’t have any default configurations similar to LXC where a bridge must be configured on the physical system and should be assigned to the containers for network access. They state that LXC and OpenVZ lead to 100% CPU usage during the initial phase, but LXC has faster initialization which minimizes the use of processors [48]. LXC can be installed directly on the Linux distributors through command line but for OpenVZ, a separate kernel has to be installed. Booting to the OpenVZ kernel allows the user to use facilities provided by the hypervisor for creating and managing containers. OpenVZ and LXC disconnection times are similar but for reconnection times LXC has an advantage over OpenVZ [48]. The issue with OpenVZ regarding reconnection times has been faced in the present study while increasing the load of the container.
through Cassandra-stress tool. The graphs of OpenVZ clearly show the deflections during increasing the load of the container at source server where it loses its connection with the Cassandra NoSQL database and takes some time to reconnect. In LXC the connections with the Cassandra-stress tool are quite constant due to quick reconnections in the case of failure which resulted in minor deflections.

In the literature [21], CPU consumption in OpenVZ is high because CPU is shared among the containers and a common kernel is used by the host and the container. The existing results show that OpenVZ has higher CPU consumption during live migration and lower migration time, but the downtime is comparatively greater than XEN-PV and XEN-HVM [21]. The present study proved that OpenVZ has lower CPU consumption than LXC and has higher total migration time and downtime.

Networking virtualization mechanism is explained in [55] where OpenVZ creates a virtual network interface for a container or Virtual Private Server (VPS) and assigns an IP address to the host system. When a packet arrives at the host system with the IP address of the container they are redirected to the container to the corresponding VPS. This approach was helpful in the present research to connect the Cassandra-stress tool to the Cassandra NoSQL database running inside the container through IP packet forwarding.

### 7.1 Independent and Dependent Variables

Independent variables do not depend on other variable i.e. the value of the independent variable is not changed by altering the values of other variables. The choice of hypervisor is an independent variable in the present study as the procedure followed and the metrics used for measuring the performance would be same for any hypervisor.

Dependent variables rely on other variables i.e. the value of the dependent variable is also changed if other variables are altered. The performance metrics, CPU utilization, Disk utilization, Load average, Downtime and Migration Time, values depend on the load of the container which is considered in this study. If the load of the container is changed, then all the values of the metrics would also change. So, the experiment is conducted for different loads to efficiently measure the performance of the hypervisors.

### 7.2 Threats to Validity

- **Internal Validity** refers to how well the experiment is conducted and how the treatment followed caused the outcome [56]. CPU load is considered as a major factor in this study. The potential threat would occur if the factors other than CPU load affect the performance of live migration. The threat is mitigated as the main intention of live migration is to balance the load among the servers, so the experiment is conducted for different loads to measure the performance of hypervisors during migration of containers. Another threat would be the effect of other processes in the server during the process of live migration. This risk is mitigated by killing all the remaining processes before conducting the experiment to ensure that the obtained results are exclusively due to live migration of containers.

- **External Validity** concerns whether the results of the experiment can be generalized to the outside population [56]. There would be a threat to the
present study if the hypervisor chosen are not been used by the cloud providers as they are outdated. The risk is mitigated as the hypervisors chosen are characteristic implementations of Linux distributors with few similarities, which are the latest and stable technologies that are widely used by the cloud providers. Another threat that would occur if the results of the study do not reflect the real world problem. The risk is mitigated by using Cassandra NoSQL database to increase the load of the container in a real world scenario where the load is varied dynamically.

- **Construct Validity** refers to the relation between the theory behind the experiment and the outcomes [56]. The metrics, Downtime and Total Migration Time, chosen for performance measurement of hypervisors during live migration of containers might not be adequate. The risk was mitigated by considering additional metrics CPU utilization, Disk utilization, and Load average to efficiently measure the performance.

- **Conclusion Validity** concerns how the treatment followed in the experiment is related to the outcome [56]. The risk is mitigated by performing the experiment repeated times and the results are considered by taking the means and standard deviations of the obtained results.
8 CONCLUSION AND FUTURE WORK

8.1 Conclusions:

The main aim of this research is to compare the performance of container-based virtualization during the process of live migration. Hypervisors chosen for this research are OpenVZ and LXC. Live migration is performed to ensure high availability of servers during maintenance and hardware failures by migrating the container to another server. Downtime and Total Migration Time are used to measure the performance of live migration. CPU utilization, Disk utilization, and Load average are also considered for the performance comparison of hypervisors. Experiment method has been used to perform live migration of OpenVZ and LXC containers. Graphs are illustrated from the results and later analyzed to find the differences between the hypervisors based on CPU utilization, Disk utilization and Load average during the process of live migration. Along with these parameters, Downtime and Total Migration Time of the Hypervisors during live migration are calculated using ping utility. A boxplot is used for the metrics, Downtime and Total Migration Time, to find the difference between the hypervisors.

From the results obtained it can be concluded that OpenVZ has poor performance regarding downtime and total migration time when compared to LXC. However, LXC has higher CPU utilization when compared to OpenVZ and has poor performance when the CPU load is high. OpenVZ is a stable hypervisor which can migrate container with any CPU load while LXC is latest and developing technology.

8.1.1 Answers to Research Questions

**RQ1:** How is live migration performed in OpenVZ and LXC containers?

**Answer:** Live migration of LXC containers is done using LXD and CRIU, which enables checkpoint restore utility to the containers. The state of the container is saved, or checkpoint is created with all the objects in the containers. This state is copied to the destination node through remote images and then the container is restored to the target server from the saved images. Successful completion of migration kills the container data on the source server.

Live migration of OpenVZ containers is performed by initially generating a private key at the source and sent to the destination key. This key is added to the authorized keys of the target server to support live migration. Once the connection is established, the container can be migrated to the destination. The first synchronization is done where container file system is moved to the destination. Suspend the resources and dump them to a file. The second synchronization is performed to update the memory pages in the container then the dump file is transferred to the destination. The container is created at the destination from the dump and is resumed with the same state. On the successful migration of the container, the data is destroyed at the source.

**RQ2:** What is the difference in performance between OpenVZ and LXC during Live migration in terms of CPU utilization, Disk utilization, Downtime and total migration time?

**Answer:** The results show that LXC has higher CPU utilization and so higher CPU load, which represents lower performance. However, LXC has the best performance when the CPU load is low as it has lower Downtime and Migration Time when
compared to OpenVZ. OpenVZ has better performance when the CPU load is high as LXC sometimes fails to migrate overloaded container.

During Live migration, the CPU Utilization and Load Average of OpenVZ increases gradually for the source and destination server while for LXC, it decreases at the source and increases at the target. Disk Utilization increases for both the hypervisors during live migration.

### 8.2 Future Work

This research work is focused on performance comparison of container-based virtualization, OpenVZ, and LXC during live migration. Possible scope for future work to the present study is as follows.

- **OpenVZ** is a stable hypervisor which can migrate container with any CPU load while LXC is a recent technology which can be developed. Further research can be done on LXC to develop algorithms to reduce CPU Utilization and permit smooth migration of highly loaded containers.

- Other metrics include total network traffic, which is the amount of data that is required to be transmitted during migration, and energy consumption, which is the extra energy used during the migration process [39]. These metrics can be analyzed for the present research work.

- Dockers were not considered in this research work as live migration is not yet available officially by the developers. Live migration in Dockers can be performed manually by using checkpoint restore feature. In future, comparing Dockers with the chosen hypervisors in the present study would be beneficial for the cloud providers.
REFERENCES


[44] “LXD 2.0: Live migration [9/12] | Stéphane Graber’s website.”.


APPENDIX A

Figure 38: Load Average for OpenVZ at Source (threads = 150)

Figure 39: Load average for OpenVZ at destination (threads = 150)
Figure 40: Literature review

Figure 41: Results for OpenVZ (threads = 150)
Figure 42: Results for OpenVZ (threads = 450)

Figure 43: Results for LXC (threads = 150)

Figure 44: Results for LXC (threads = 450)
APPENDIX B

LXC setup and migration steps

- sudo apt install lxc
- sudo apt install criu
- sudo apt install lxd
- sudo lxd init
- lxc config set core.https_address [:]
- lxc config set core.trust_password some-password
- lxc config set images.remote_cache_expiry 5
- lxc config set images.auto_update_interval 24
- lxc config set images.auto_update Cached false
- lxc launch ubuntu: migratee
- lxc profile create migratable
- lxc profile edit migratable

------------------------------------------
# The configurations settings to be edited in the file are as follows
name: migratable
config:
raw.lxc: |
  lxc.console = none
  lxc.cgroup.devices.deny = c 5:1 rwm
  lxc.start.auto =
  lxc.start.auto = proc:mixed sys:mixed
  security.privileged: "true"
devices:
  eth0:
    nictype: bridged
    parent: lxcb0
    type: nic

------------------------------------------
- lxc profile apply migratee migratable
- lxc remote add lxd this_ip:port
- lxc remote add lxd2 other_ip:port
- lxc profile copy migratable lxd2:
- lxc profile list
- lxc profile show migratable
- lxc move lxd:migratee lxd2:migratee
OpenVZ

Installation:

- Download openvz.repo file and put it to your /etc/yum.repos.d/ repository:
  
  ```
  wget -P /etc/yum.repos.d/ https://download.openvz.org/openvz.repo
  ```

- Import OpenVZ GPG key used for signing RPM packages:
  
  ```
  rpm --import http://download.openvz.org/RPM-GPG-Key-OpenVZ
  ```

- Download latest vzkernel, vzctl, plop and vzquota rpm packages:
  
  ```
  cd /tmp
  ```

- Install following packages
  
  ```
  apt-get install fakeroot alien libcgroup1
  ```

- Convert RPM to deb packages
  
  ```
  fakeroot alien --to-deb --scripts --keep-version vz*.rpm ploop*.rpm
  ```

- Install deb files
  
  ```
  dpkg -i vz*.deb ploop*.deb
  ```

- There are a number of kernel parameters that should be set for OpenVZ to work correctly. These parameters are stored in /etc/sysctl.conf file. The necessary changes to the parameters are shown in the Figure 41.
Figure 45: Configuration settings for OpenVZ

Creation of containers:

- Vzctl create 101 –ostemplate ubuntu-14.04-x86_64
- Vzctl set 101 --nameserver " " --save
- Vzctl set 101 --ipadd " " --save
- Vzctl start 101

Live Migration:

- cd /usr/local/bin
- wget http://files.soluslabs.com/solusvm/scripts/keyput.sh
- chmod a+x keyput.sh
- nano /etc/ssh/sshd_config
  
  change #PasswordAuthentication yes to PasswordAuthentication no
  and #PermitRootLogin without-password to PermitRootLogin yes

- ./keyput.sh destination_node_ip (or) ssh-keyput destination_ip
- vzmigrate –online destination_node container_id

```plaintext
# On Hardware Node we generally need
# packet forwarding enabled and proxy arp disabled

net.ipv4.ip_forward = 1
net.ipv6.conf.default.forwarding = 1
net.ipv6.conf.all.forwarding = 1
net.ipv4.conf.default.proxy_arp = 0

# Enables source route verification
net.ipv4.conf.all_rp_filter = 1

# Enables the magic-sysrq key
kernel.sysrq = 1

# We do not want all our interfaces to send redirects
net.ipv4.conf.default.send_redirects = 1
net.ipv4.conf.all.send_redirects = 0
```