Performance Analysis of Self Organized Load Balancer Webservers over Virtualization and Dockerization Technologies.

Submitted by:

Muhammad Shakeeb

00000118198

Supervised by:

Dr. Bilal Muhammad Khan

THESIS

Submitted to:

Department of Electronic and Power Engineering
Pakistan Navy Engineering College, Karachi
National University of Sciences and Technology, Islamabad, Pakistan

In partial fulfillment of requirements for the award of the degree of
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
Specialization in Communication

FORM TH-4

National University of Sciences and Technology

MASTER'S THESIS WORK

We hereby recommend that the dissertation prepared under our supervision by: (Student Name & Regn No.) Muhammad Shakeeb & Regn No. 00000118198

Titled: **Performance Analysis of Self Organized Load Balancer Webservers over Virtualization and Dockerization Technologies** be accepted in partial fulfillment of the requirements for the award of Masters degree.

Examination Committee Members

1. Name : Dr Syed Sajjad Haider Zaidi Signature : [illegible]

2. Name : Dr Ali Hanzala Khan Signature : [illegible]

Supervisor's name: Dr Bilal Muhammad Khan Signature: [illegible]

Date: 31-10-2018

[illegible]
DR. ATTAULLAH MEMON
Captain Pakistan Navy
Dean Faculty of Electronics &
Power Engineering
NUST-PNEC
Head of Department

19 Nov 18
Date

COUNTERSIGNED

Date: 20-11-2018

[illegible]
HABIB UR REHMAN SI (M)
Commodore
Deputy Commandant
PNS JAUHAR
Dean / Principal

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS thesis written by Muhammad Shakeeb & Regn No. 00000118198 of PNEC (College) has been vetted by undersigned, found complete in all respects as per NUST Status/Regulations, is free of plagiarism, errors, and mistakes and is accepted as partial fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have been incorporated in the said thesis.

Signature: [illegible]

Name of Supervisor Dr Bilal Muhammad Khan

Dated: 31-Oct-2018

Signature: HoD [illegible]

Dated: 05 Nov 18

Signature: (Dean/Principal): [illegible] HABIB UR REHMAN SI (M)
Commodore
Deputy Commandant
PNS JAUHAR

Dated: 05-11-18

بسم الله الرحمن الرحيم

ACKNOWLEDGMENTS

All praise and thanks to the Almighty ALLAH (SWT) whom we seek help and guidance for sustenance. I would like to express sincere gratitude to my advisor Dr. Bilal Muhammad Khan for his patience, motivation, and continuous support for my Master’s study and his wise reproach to complete related research. His timely guidance helped me in writing of this thesis. He helps me with his insight knowledge and enlightening me the first glance of my research, understanding the subject and facilitating with in completing this thesis. I could not have imagined having a better mentor and advisor for my Masters than him hence, highly grateful to him for his kind support in all phases of research. Besides my advisor, I would like to thank and appreciate my thesis GEC members: Dr Ali Hanzala Khan and Dr Sajjad Haider Zaidi for their insightful encouragement and comments, but also for the question which incented me to widen my research approach from various perspectives and implementation. Last but not the least, I would like to thank my family: my parents, brother and my sister for supporting me spiritually throughout my research and writing this thesis.

ABSTRACT

Self-organize and load balancing of computing system is one of the emerging research area. Despite of having robust infrastructure; several organizations still face system downtime due to hardware failure, high load, memory exhaustion and network choking. The best way to avoid downtime is to have the ability to predict and analyze the behavior of link and infrastructure. This paper presents an experimental implementation and evaluation of load balancing using HA-Proxy as a software-based load balancer and a proactive approach for self-organizing of web systems using certain algorithms in case of system failure over virtualization. We are using Network function virtualization using XenServer that provides the ability to take network and traffic management. Furthermore designing a system using Docker container technology a containerized software which runs as same regardless of the environment and analysis its response time with respect to virtual technology. However, making sure that achieving such centralized system is not a trivial task as it raises various issues relating to network control, system management, data security, package requirements, routing and data migration.

Table of Content

FIGURES

TABLES

LIST OF ACRONYMS

Abbreviation	Meaning
VM	Virtual Machine
VMM	Virtual Machine Monitor
LVS	Linux Virtual System
HA	High Availability
RAM	Random Access Memory
UML	User-mode Line
CaaS	Containers as a Service
PHP	Hypertext Preprocessor
CMS	Content Management System
SaaS	Software as a Service
CLI	Command Line Interface
SCP	Secure Copy
LXC	Linux Containers
lmctfy	Let Me Contain That For You (virtualization)
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
CPU	Central Processing Unit

BSD	Berkeley Software Distribution
API	Application program interface
PaaS	Platform as a Service
IT	Information Technology
OS	Operating System
NIC	Network Interface card
ab	Apache bench
MB	Megabytes
ms	Milliseconds
MHz	Megahertz
MiB	Mebibyte
cgroups	Control groups
TCP	Transmission Control Protocol
UDP	User datagram protocol
IaaS	Infrastructure as a Service

PUBLICATIONS

Refereed Conference Papers

Muhammad Shakeeb, and Bilal M. Khan, “EVALUATING AND EXPERIMENTAL ANALYSIS OF LOAD BALANCER AND SELF ORGANIZED WEBSERVERS USING HA-PROXY”, *3rd International Electrical Engineering Conference (IEEC 2018)*

CHAPTER 1: INTRODUCTION

1.1 Problem Statement

The main problem: Why I select this topic?

- High Web Traffic causes system high load
- Due to high response time customer get frustrate due to slow browsing
- Due to congestion and choking system failure chances may increase
- If we bought special dedicated hardware than the cost may increase.
- If any disaster arises than it is very difficult to build same exact environment and it causes high down time

1.2 Overview

World is a ground of daily happening disasters and in past some decades global climate change increases the severity of both Human, Infrastructure and Economic losses. Disasters either natural, environmental or manmade can lead to an expensive service disruption. The key obstacle to any response is communication and collaboration support therefore provisioning of real time data to base station is always needed. Communication System in any location always effected by incidents while the occurrence of incident cannot be predicted and its impact can widely be felt due to its high utilization in such conditions either accessing internet, standard mobile phones, landline telecommunication and even satellite-based communication devices which is critical to survival, security and safety of the world.

"When Disaster arise, Communication saves life "

Communication System is the key for efficient business operations which provides path and services between users, applications, processes. Due to high reliability and automation most of the organizations are adopting different technologies and certified standards to keep their infrastructure reliable and ensure system continuity to run business smooth. Several organizations run business 24 / 7 and due to these continued operations enterprise solutions is required to determine its ability to deal with natural, potential or man-made errors. These solutions helps through creating an effective disaster recovery plan that can enable minimizing disruptions to the infrastructure, and quickly restore operations to normalcy. Noting extensive and potential benefits many of researchers, scientist and Engineers are working to build algorithms and develop scalable system for on time organizing and healing of network. Despite of having robust infrastructure; several organizations still face system downtime due to hardware failure, high load, memory exhaustion and network choking.

1.3 General Background

The development of an intelligent system is not a trivial when life threatening situation may involve therefore researchers are working to develop Self-organized system to up this world. Load balancing of a system is a way to effectively handle traffic where there is high volume of users accessing the system, information delivery and shift traffic to such location where there is service availability is normal. It also helps to decrease on site recovery, reducing load times and accessibility problems. By properly and evenly distributing traffic to more than one system organizations can improve response time and network throughput.

1.4 Thesis Approach

To the best of knowledge, there is not much research carried out in the field of communication especially related to Cloud Computing and Self-Organized web servers in Pakistan and this is one of reason of working in this domain. It is also important to note that such system which is responsible to handle high traffic when required is in demand for most of the organizations who don't want to invest much on infrastructure and trying to reduce cost. The proposed thesis would try to overcome these issues and provide a suitable solution that can work efficiently with less human intervention and gives better output.

Initially, we discuss about Load Balancer and its different algorithms using HA-Proxy. The purpose behind that study was to know the reason that which algorithm is most suitable whose response time is low over Virtual machines or Docker as a Container Technology and why. After System Load Balancer design, a simulations for self-organize Web Servers and Containers.

1.5 Thesis Challenge:

The major challenge of proposed thesis analyze the behavior of Single Host and multiple host over load balancer and to check the response time after catering all client request. 2nd major challenge is to develop such environment where main host predicts the behavior of the traffic and generates alert if there is high system memory / load after that system organize itself automatically without human intervention by adding extra host to balance the traffic and system utilization.

1.6 Thesis Organization

The study comprises of the following sections.

- Chapter 2 consists of an introduction to hypervisor virtualization technology and Container. It also presents a detailed introduction, of HA-Proxy with Load Balancing algorithms.
- Chapter 3 deals with the Experiment analysis of Load balancer and Self-Organize system over virtual machines.
- Chapter 4 discusses about Docker container and evaluate load balancer over it. Similarly develop an algorithm to organize container if there is high load.
- Chapter 5 finally, concludes experimental results and propose the future work directions.

CHAPTER 2: LITERATURE REVIEW

Following chapter provides an insight on the fundamental concepts of Load Balancer. The chapter also discusses the terminology Hypervisor with Virtualization and Container. The chapter also explains various algorithm of HA-Proxy as Load Balancer.

2.1. VIRTUALIZATION

Virtualization is the most effective technology to work with in the distributed environment and to feel exact real environment these virtualization is developed [1]. For the system development it embeds a preproduction site and offers developer machines a real runtime system development environment [2], [3]. Thus, this technology allowing dynamic allocation of software and hardware and made system development lifecycle flexible. To build, test and integrate a complex system virtualized environment is always needed [4], [5], and [6]. However, to setup such virtualized environment it requires skilled IT staff having knowledge and good understanding to build, installation and configuration [7] and keeping in mind about wide range of technologies including its operating systems, application, infrastructure and network services. Therefore virtualization makes integration much harder into production environment [1].The biggest issues with the virtualization despite of managing all is resource utilization and operational overheads, which makes it less adoptable [8]. Virtual Machine may act as a logical server and can be hosted in physical machines which have complete isolation from the main server as it runs its own instances. Each virtual machine act as a real machine having its own tools, software’s, users and network configuration. Virtualization allows users to create virtual environment of a resource, Such as storage devices, operating System, network firewalls and private cloud servers. Virtualization is the combination of hardware and software that creates an abstraction layer that enables multiple operating systems to run on (single computer hardware) same physical platform [24]. Virtualization framework helps system resource to divide it in one or more executable environments [9]

Architecture of Virtualization has two major components:

- VMM a Virtual Machine Monitor also called Hypervisor
- VM a Virtual Machine where you install the operating system

The virtual machine monitor is a software responsible to run VM also called guest operating system or a guest computer. It controls and responsible to translate system hardware and virtual machine within the virtualization technology [10]. These can be classified in two types: A Bare metal hypervisor which is also knows as Type 1 hypervisor that runs directly on the hardware of host without the need of OS on a host. Second, hypervisor a Type 2 runs on top of host operating system and loads higher level VM.

A Virtual Machine is like an independent operating system that behaves as a separate computer [21] generally, every VM has its own environment, operation system, applications, libraries, utilities and system binaries. Virtualization usually requires storage, processing capacities and bandwidth.

A large system is required, if multiple host need to be run [11]. The reason of introducing virtualization in system is the features provided by it which has attracted the attention of both academic and industrial bodies. Several works have been done on both system virtualization and virtualized networks [10]. The technologies available such as VMWare, Virtual Box, UML and Xen. VMMs includes VMware server/ESX server [29], User Mode Line [30] and Citrix Xen Server [30, 25]. VMware provides virtualization for both Linux and windows based architecture. VMware is a well-known commercial product used in production environment. Para virtualization present a modified interface having a new architecture for the interface of guest OS. Xen is a para virtualized virtual machine monitor that supports Berkeley Software Distribution (BSD) and a Linux based OS. UML also supports virtualization but have low speed than Xen VMM; therefore, in this paper XenServer is used which is configured on Physical Machine and to monitor it we will use Xen virtual machine monitoring app. In the rest of the research, VM and Virtualization terminology represents Type 2 hypervisor and this category as architecture is comparable to Dockerization.

2.2. DOCKERIZATION

Linux based operating system offers portable, lightweight and high performance containers as compare to virtualization. Linux kernel has a feature called Control Group which helps to isolate the resource usage and name space that allows independent containers to run with in single OS. The cgroups is used by the Docker as isolation feature helps in avoiding overhead of start and stopping VMs. UNIX introduced chroot command in the year of 1979. Later in the year of 1988, jail is introduces by FreeBSD that extends from chroot. After few years Solaris 10 introduces zones which extends the capabilities of chroot and these zones extends further to become Containers in Sun Solaris 11.

Linux introduces LXC that uses user space interface as containment functionality for Linux Kernel and can be easily manage through API and CLI tools [35]. Each application packed with of independent network, memory, files, binaries and system libraries which act as an independent container. The Container Engine manages these containers as they are extent from OS Kernel and share same OS in a fraction of size and allowing hundreds of containers within a single OS. The container can be deployed in two modes either LXC or as a Docker.

Docker a container technology designed to run a single application per container. LXC offers containers that are lightweight and work like a virtual machines where user may login and install its own application. Container is known with OS-level virtualization [19] because virtualization user space is on top of OS kernel. These containers allow multiple isolated user space system to run on a single host [19]. Popular containers technologies available such as OpenVZ and systemd-nspawn, LXC, lmctfy, Warden [19], [20].

An open source engine Docker is developed by Docker Inc formerly known dotCloud. Docker is an example of container virtualization and its process known as Dockerization. Linux Kernel feature cgroups and namespace is the base of Docker and its engine helps to automate the deployment of applications into Containers [19]. Two major classification of these containers are as below:

- Complete operating system which runs all processes like: sshd, init, cron and syslog is a categorized as System Container
- Containers which runs on application is categorized under Application Container [20].

These containers can be used under different circumtances. Each container is completely isolated and having own file system, storage, stack, networking, resource utilization and its management which helps to run multiple containers on a single host [20].

2.3. Virtual Machine Vs Docker Container

Cloud platforms such as Amazon, Digital Ocean, Vultr, and Google Cloud Engine makes virtual machines runs many platform, software and infrastructure related services which are available for the customers to use for their desired requirements. PaaS and SaaS providers are built on IaaS builds on virtual machines and used by cloud services therefore it is very crucial component to manage the workloads on these virtual machines.

In cloud computing container presents an alternative solution to these virtual machines, these Docker a container based virtualization is used for faster functionalities [37].As compare to virtual machines, start and stop the container is much faster.

Virtualization was born out of need to use multiple OSs on a single hardware and main motivation behind is was to reduce cost of maintaining hardware servers and take economic benefits of running multiple isolated workloads on a single machine. Whereas Docker was born out to of need to abstract out an application's encompassing eco system (OS, Network Stack, CPU, Software modules etc.) so that an application runs in a predictable way in all the environments (Development

/ Staging / Production etc.). Container Technology popularity increases due to a lot of factors. As any complex application can be containerized so it is very Flexible, similarly you can update upgrade the deployments so simply called having functionality of Interchangeable.   Further Table No.1 summarizes basic difference between Docker container and these virtual machines.

Table 1: Difference between VM and Dockers

S.No.	Virtual Machine	Docker Container
1	Host virtual machine runs on a physical machine and guest operating system is loaded in its memory.	Guest's shares same host operating system, This guest is loaded in host physical memory.
2	Guests and host can communicate through network devices, may be software.	Communication between host and guests is through sockets, bridges network and pipes.
3	More overhead due to its complexity.	it is light weight so a less overhead
4	Security depends on the hypervisor.	Lacks security measures
5	Libraries and files are not possible to share in virtual machines	File sharing is possible in Docker container either using SCP or Linux utilities
6	Booting of operating system takes much time	Faster booting.
7	Uses much memory as it needs to store complete operating system of each guests.	Less memory usage as it shared host's operating system

2.4. Load Balancer with HA-Proxy

Social media popular websites like Facebook, twitter etc. have a huge number of users and need to be served in real time which causes high load on server due to server limitations response time increases so instantly some time causes server downtime. Number of factors is responsible for service degradation and sometimes leads to system failures. To maintain the service and up response, multiple servers must be provided to prevent bottlenecks and allowed to forward the client requests to the backend servers to handle them which helps to full fill all client request. To handle this a load balancing system is required. Web Load balancer receives user requests and is responsible to distributes to multiple defined backend servers. Load Balancing of network, web can be implemented with either dedicated hardware, software or a combination of hardware and software both. HA-Proxy is a well-known software based load balancer used with Linux Operating system. In IT industry high availability refers to a system that is always available and in active operational mode for continuous delivery. Different layers works together to provide service availability [22]. User requests are dispatched using load balancing policies defined in load balancer to end servers. Cluster of Load Balancer consists of the main load balancer server and a server farm which is behind this load balancer

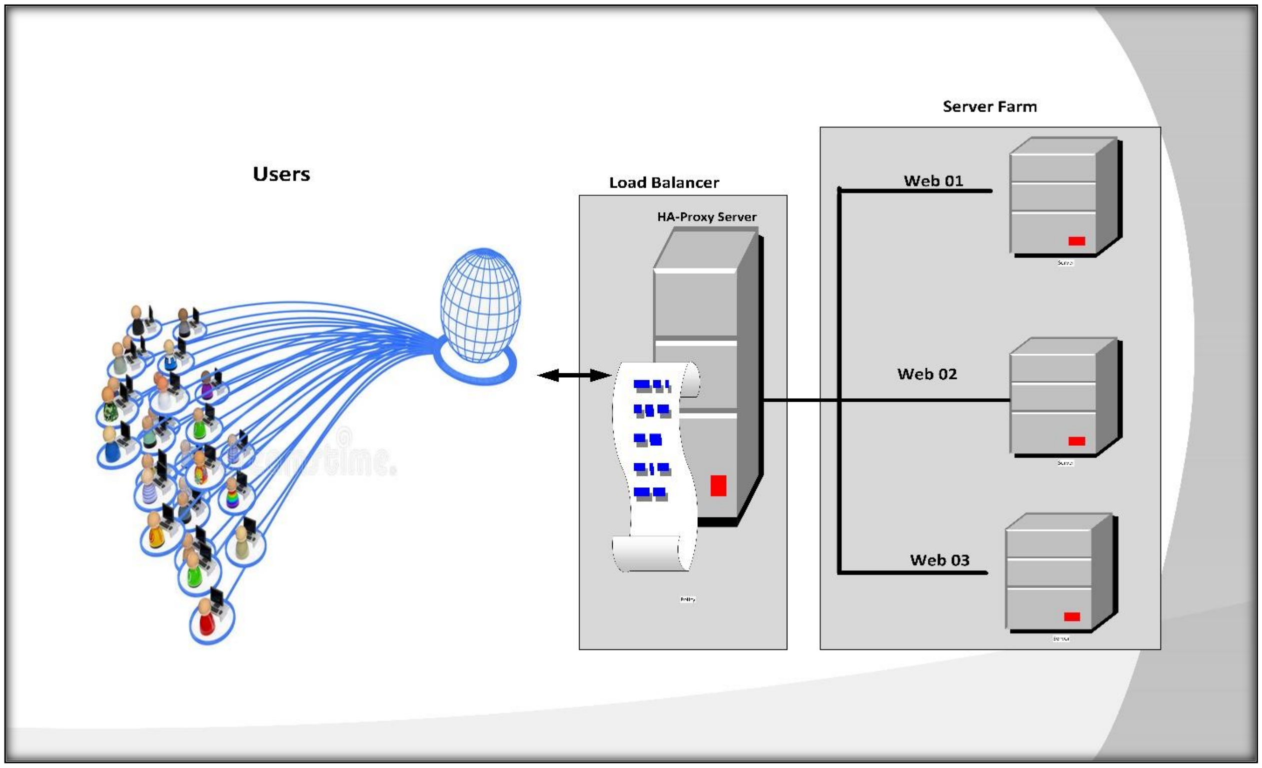

Figure 1: Load Balancer and Server Farm

Load balancer distributes user’s traffic to one of the server which is available in zone defined in the configuration. Vendors like Cisco makes hardware devices which is dedicatedly designed to handle the users request as fast and efficiently as possible [23]. In this research HA-Proxy an open source Linux based load balancer is used.

2.5. Algorithms used with HA-Proxy Load Balancer

Load balancer using HA-Proxy uses a number of algorithms whose aim is to minimize the imbalance between different servers and distribute the load among in an optimal way. Load balancer supports several algorithm which includes: first alive, hash, round robin, least connections, weighted least connection and weighted round robin. These all are different policies used with respect to the system requirement.

2.5.1. First alive

This algorithm uses primary and secondary as a backup server concept. First server defined in the member list in HA-Proxy configuration is the Primary server similarly the secondary servers are next servers in the list. First alive algorithm checks health state of first server if it is up it will forward all the traffic to primary server and if the health state of the primary server is down, the data power service forward all connections to the subsequent servers.

2.5.2. Hash

HTTP header and the IP address of the user is the basis for server selection in this algorithm. This feature is only available for web Service proxy and Multi-Protocol Gateway. Hashing algorithms cannot ensure evenly distributed connections. [9]

2.5.3. Least connections

This algorithm stored a detail of all the online servers and its connections and is responsible to forward new connection request to the server which has less active connections. [9]

2.5.4. Weighted least connections

This type of algorithm helps to maintain and generate the list of application servers which has active numbers of user connections. On the basis of the number of active connections new connections request is forwarded to desired servers. It utilizes much computation time so increasing response time therefore known as slow algorithm.

2.5.5. Round robin

Round robin algorithm is known with its name as it is depends on the list of servers defined in HA-Proxy configuration. This algorithm utilizes the concept of forwarding a new client connection request to the next backend server in configuration list [9].

These algorithms can work along with more feature either using with weighted or non-weighted. In weighted it checks server with higher weights and sent the request to that server who has a higher weight. In non-weighted this algorithm assumes the capacity of existing servers defined in group are equivalent so transfer all request equally to all servers therefore it usually known for faster than weighted algorithms.

CHAPTER 3: EXPERIMENTAL ANALYSIS OF LOADBALANCER WEBSERVERS OVER VIRTUALIZATION

This chapter discusses in detail the technology used for virtualization hypervisor, to support service availability a load balancer is needed which requires two or more servers. A load balancer receives traffic from external sources and distributes these requests to multiple defined load balancer servers. Our design consists of two virtual webservers and on top of it a load balancer.

The ability to run such infrastructure having elements VMs and a Virtual machine monitor, Several software appliances around for a long time that supports to view, monitor and manage multiple virtual machines with a single host. In this research Xen an open source Virtualization Technology is used to host Linux OS [25]. Network performance, measurement and quality of a service can be achieved by the hardware probes of the system. However special configuration of network monitoring system is needed that runs on all gateway services on single physical machine. Usually hardware probe work around system clock and reports it to monitoring system with these probe results in real time in order to fully maintain quality of service and system production [26].

In any communication environment availability of the network is highly required. This performance factor of network availability can be determined by analyzing the connectivity of end-to-end process flow. To test network connectivity ARP and ICMP packets are sent to test and diagnose the network connectivity [27]. The aim of the proposed algorithm is to gather information from multiple sources using network utilities and later on using algorithm including from the point of incident and make effective decisions including shifting of the network traffic from one VM to other VM in case of any disaster or system degradation. Moreover, the proposed algorithm provides a unique solution and work on all such physical machine [28].

3.1. System Design

System configuration may vary according to the hosting of virtual machine. In this research three virtual machines created that is hosted on single physical machine and managed by Citrix Xen management and monitoring tool.

3.1.1. Xen Server

To implement network virtualization at least two separate physical machines required one having Xen Server installed, and the other to run the Xen monitoring tool i.e. Xen App. This paper uses a system with multiple core enabled with 64bit server-class machine to allow virtualization [32] [25]. Xen machine has xen-enabled kernel with an optimized Linux partition which is responsible to controls the interaction and its communication between virtual devices and the physical hardware [33].

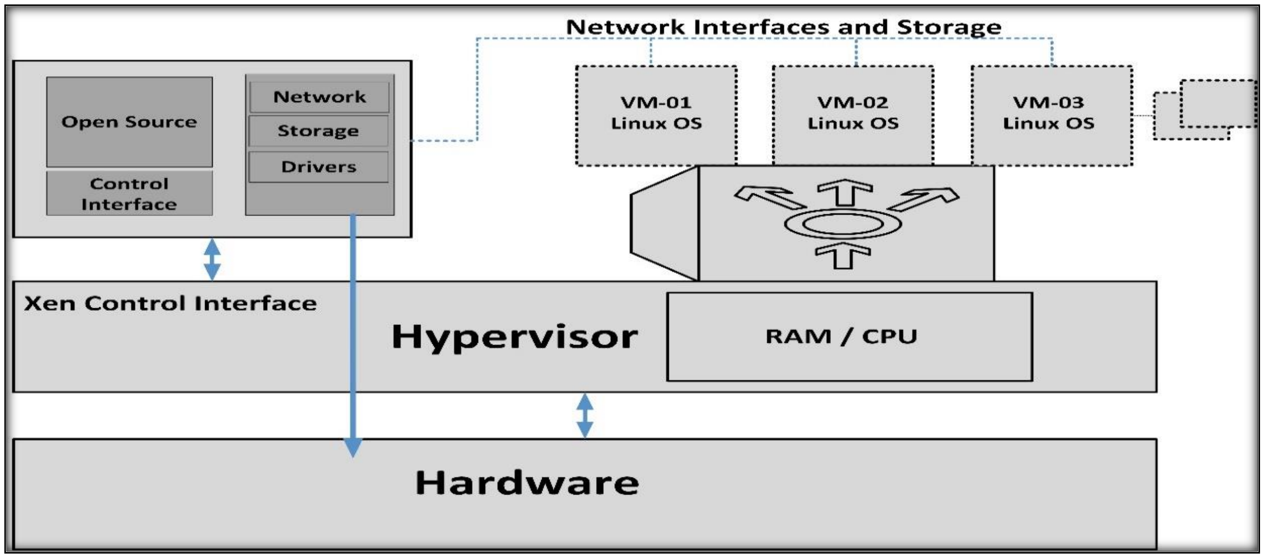

Figure 2: XEN Architecture

3.1.2. Xen Remote Management

Xen App is installed on any supported operating system. It enables secure, remote access of Xen Hosted server. Through Citrix Xen app a remote management tool is used to install the virtual machine in Xen Hosted Server [34].

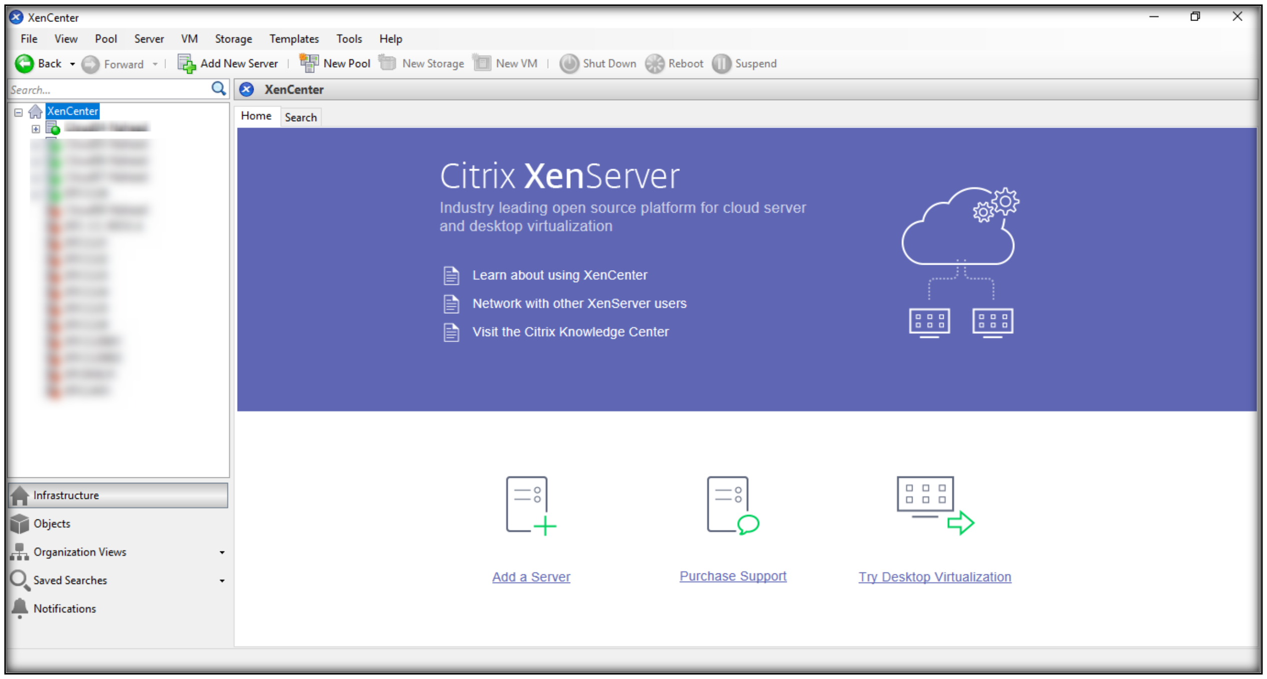
XenCenter
File View Pool Server VM Storage Templates Tools Help
Back Forward Add New Server New Pool New Storage New VM Shut Down Reboot Suspend
Search...
XenCenter
Home Search
Citrix XenServer
Industry leading open source platform for cloud server and desktop virtualization
Learn about using XenCenter
Network with other XenServer users
Visit the Citrix Knowledge Center
Add a Server
Purchase Support
Try Desktop Virtualization
Infrastructure
Objects
Organization Views
Saved Searches
Notifications

Figure 3: XEN Remote Management Interface

3.1.3. Xen Virtual Network

Xen enabled kernel creates virtual network, virtual interfaces like network interface cards for VMs and these Virtual network interfaces are bridge to real interface using Linux system utilities. Virtual machine has its own identity usually defined by the user which allows access from gateway machine [34] to access it remotely. We can setup a public IP as well to directly access it remotely.

Figure.No. 4 presents the internal architecture of Xen server hosted physical machine with its network interfaces.

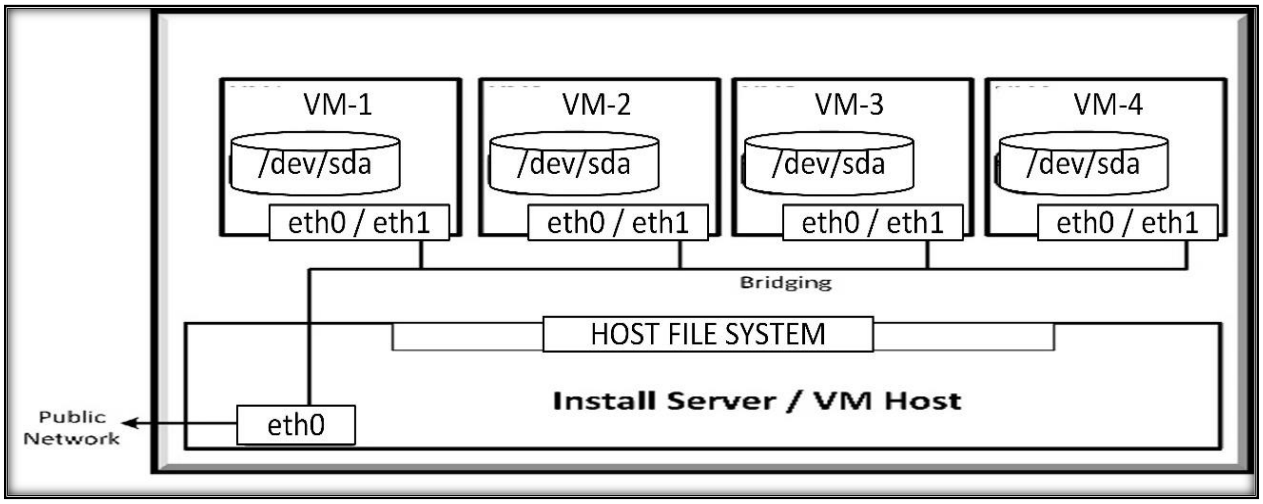

Figure 4: XEN Virtual Network Architecture

System requirements to configured virtual machine shows in Table No.2. These may be varying as per the hardware size and system traffic requirement. In this paper three VMs are used two of these VMs has 1 network interface and the main load balancer have 2 network interface cards. Ubuntu as operating system is used here with all the VMs.

Table 2: Virtual Machine Configuration Table

System Req.	Host-1	Host-2	HA-Host
Processor	2700Mhz	2700MHz	2700MHz
RAM	512MB	512MB	1024MB
Disk	6GB	6GB	6GB
NIC	1	1	2

3.2. Load Balancing Using HA-Proxy on Virtual Machine.

Load balancing as a service is offered by most of the cloud hosting providers and its demand increases so rapidly [35] [36] due to high internet usage. In this research several experiments performed to test the availability and cater client requests. Round robin is the most common among available load balancing polices, and is supported by major providers [37]. To evaluate VM performance using Load balancer we have created two apache webservers and install desired packages and system requirements. Each contains same data and apache configuration file so to analyze each with same value.

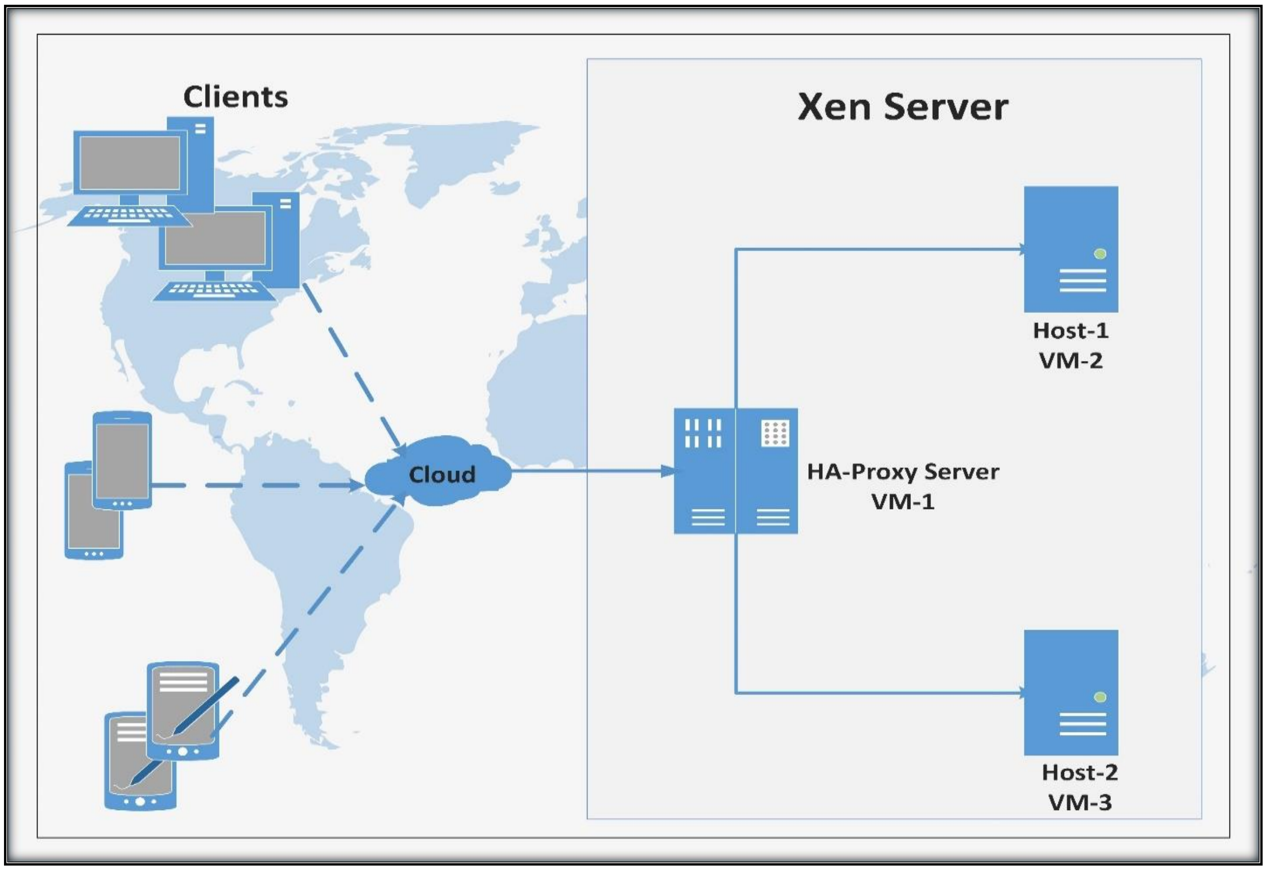

Figure 5: Load balancing using HA-Proxy

Assign static IP addresses 172.16.100.5 (host-1) and 172.16.100.6 (host-2) to both VMs. Installed HA-Proxy using below commands on Linux Main host which is responsible to distribute the traffic to desired host.

```
root@MainServer:~# apt-get install haproxy
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  vim-haproxy
The following NEW packages will be installed:
  haproxy
0 upgraded, 1 newly installed, 0 to remove and 91 not upgraded.
Need to get 0 B/417 kB of archives.
After this operation, 823 kB of additional disk space will be used.
Selecting previously unselected package haproxy.
(Reading database ... 59827 files and directories currently installed.)
Preparing to unpack .../haproxy_1.4.24-2ubuntu0.4_amd64.deb ...
Unpacking haproxy (1.4.24-2ubuntu0.4) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Processing triggers for ureadahead (0.100.0-16) ...
ureadahead will be reprofiled on next reboot
Setting up haproxy (1.4.24-2ubuntu0.4) ...
 * Starting haproxy haproxy
```

HA-proxy is used as the main server which handles all client request and forward it to these web servers with name lamp1 and lamp2 in above configuration. Assign static IP to HA-Proxy Server which is public interface and all the requests is received by this host.

Now need to update HA-Proxy configuration which resides in /etc/haproxy folder. Make sure to active port 80 as user request generated on it. Balance mode is set to Round robin as shown in below config.

```
root@MainServer:~# cat /etc/haproxy/haproxy.cfg
global
    log 127.0.0.1 local0 notice
    maxconn 2000
    user haproxy
    group haproxy
defaults
    log     global
    mode    http
    option  httplog
    option  dontlognull
    retries 3
    option redispatch
    timeout connect  5000
    timeout client  10000
    timeout server  10000

listen appname 0.0.0.0:80
    mode http
    stats enable
    stats uri /haproxy?stats
    stats realm Strictly\ Private
    stats auth A_Username:YourPassword
    stats auth Another_User:passwd
    balance roundrobin
    option httpclose
    option forwardfor
    server lamp1 172.16.100.5:80 check
    server lamp2 172.16.100.6:80 check
```

In this research Round robin algorithm is used to test the load balancer as it is most compatible algorithm with system designed here. To analyze round robin algorithm create a php based code with file name iptest.php contains parameter to pull exact IP details of user / client and server. Iptest file contains web Server IP and defined with a php code as .$_SERVER['SERVER_ADDR'] which is responsible to pull detail of backend server (for e.g. Host-1 or Host-2), Load Balancer Server IP with .$_SERVER['REMOTE_ADDR'] which pull the information of Load Balancer on which HA proxy is installed and X-Forwarded is the server environment variable helps to pull IP of the customer/user .$_SERVER['HTTP_X_FORWARDED_FOR'] which is client IP from where request is generated.

```
<?php
header('Content-Type: text/plain');
echo "Backend Server IP: ".$_SERVER['SERVER_ADDR'];
echo "\nLoad Balancer IP: ".$_SERVER['REMOTE_ADDR'];
echo "\nRequest Generated From: ".$_SERVER['HTTP_X_FORWARDED_FOR'];
?>
```

To analyze round robin algorithm curl request is generated with iptest code file. Curl is a tool to transfer data from or to a server. In this paper web content is analyz so runs below command to test the web page information over command line.

```
shakeeb@Shakeeb-Laptop:~$curl  http://172.16.100.7/iptest.php  Backend  Server
IP: 172.16.100.5
Load Balancer IP: 172.16.100.7
Request Generated From: 172.16.100.4
shakeeb@Shakeeb-Laptop:~$ curl  http://172.16.100.7/iptest.php  Backend  Server
IP: 172.16.100.6
Load Balancer IP: 172.16.100.7
Request Generated From: 172.16.100.4
shakeeb@Shakeeb-Laptop:~$ curl  http://172.16.100.7/iptest.php  Backend  Server
IP: 172.16.100.5
Load Balancer IP: 172.16.100.7
Request Generated From: 172.16.100.4
shakeeb@Shakeeb-Laptop:~$ curl  http://172.16.100.7/iptest.php  Backend  Server
IP: 172.16.100.6
Load Balancer IP: 172.16.100.7
Request Generated From: 172.16.100.4
```

Each client request has been verified by seeing that round robin algorithm works without any data loss and request has been handled by each server consecutively. First request is catered by the host-1 (Backend Server IP: 172.16.100.5) and second request by host-2 (Backend Server IP: 172.16.100.6) after that 3rd request is surfed by host-1 again and fourth by host-2 and so on. This behavior of alternate Server IP shows round robin algorithm. We are using only two webservers as a load balancer so only two IPs appears as a server IP.

3.3. Experimental Analysis with or without using Load Balancer.

Below are few experiments presented which measures performance analysis of using load balancer and using virtual machines having same system configuration with the help of apache bench test.

a) Web stress test without using multiple webservers (Host-1) a 512MB webserver.
b) Web stress test without using multiple webserver, double the memory of (Host-1) 1024MB.
c) Web stress test using multiple webserver, a load balancer using two 512MB webservers.

All experiments have been performed to check the performance on HA-Proxy server by increasing number of concurrent connection from 1user to 100users and then analyze response time for each request. These results based on number of connections, response time, connection rate, and timeouts. In this research a demo webpage is used and can be downloaded easily from https://www.free-css.com/assets/files/free-css-templates/download/page220/appsea.zip with a size of 1,130,489 bytes so that to analyze same page over all presented scenarios.

Apache bench (ab) is a well know web stress test tool which is used by most of the engineers / software architects to test their websites.

The parameter defined as below:

-*e (Print the result and save it in define file)*

-*n (Number of Test Perform)*

-*c (Number of Concurrent Users)*

Below is the sample shows result is store in a file called 10users, number of test performed is 100 and 10 concurrent users. Each test shows minimum response time, max response time for this particular apache bench stress test.

```
shakeeb@Shakeeb-Laptop:/mnt/c/Users/goto$  ab  -e  10users  -n  100  -c  10
http://172.16.100.7/appsea
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
Benchmarking 172.16.100.7 (be patient).....done
Server Software:        Apache/2.4.10
Server Hostname:        172.16.100.7
Server Port:            80

Document Path:          /appsea
Document Length:        313 bytes

Concurrency Level:      10
Time taken for tests:   0.252 seconds
Complete requests:      100
Failed requests:        0
Non-2xx responses:      100
Total transferred:      54000 bytes
HTML transferred:       31300 bytes
Requests per second:    397.15 [#/sec] (mean)
Time per request:       25.179 [ms] (mean)
Time per request:       2.518 [ms] (mean, across all concurrent requests)
Transfer rate:          209.43 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   0.6      1       4
Processing:    10   23   5.0     23      34
Waiting:        6   20   6.2     21      34
Total:         11   24   5.1     25      35
Percentage of the requests served within a certain time (ms)
  50%     25
  66%     26
  75%     28
  80%     28
  90%     30
  95%     32
  98%     35
  99%     35
  100%     35 (longest request)
```

3.3.1. Web stress test using single webservers (Host-1)

First Scenario has been tested by switching off all the web server except (Host-1) here host-1 has a memory of 512MB and to analyze its behavior a curl request is generated. Removing all the IPs from ha-proxy configuration so that only single host is responsible to handle all client request.

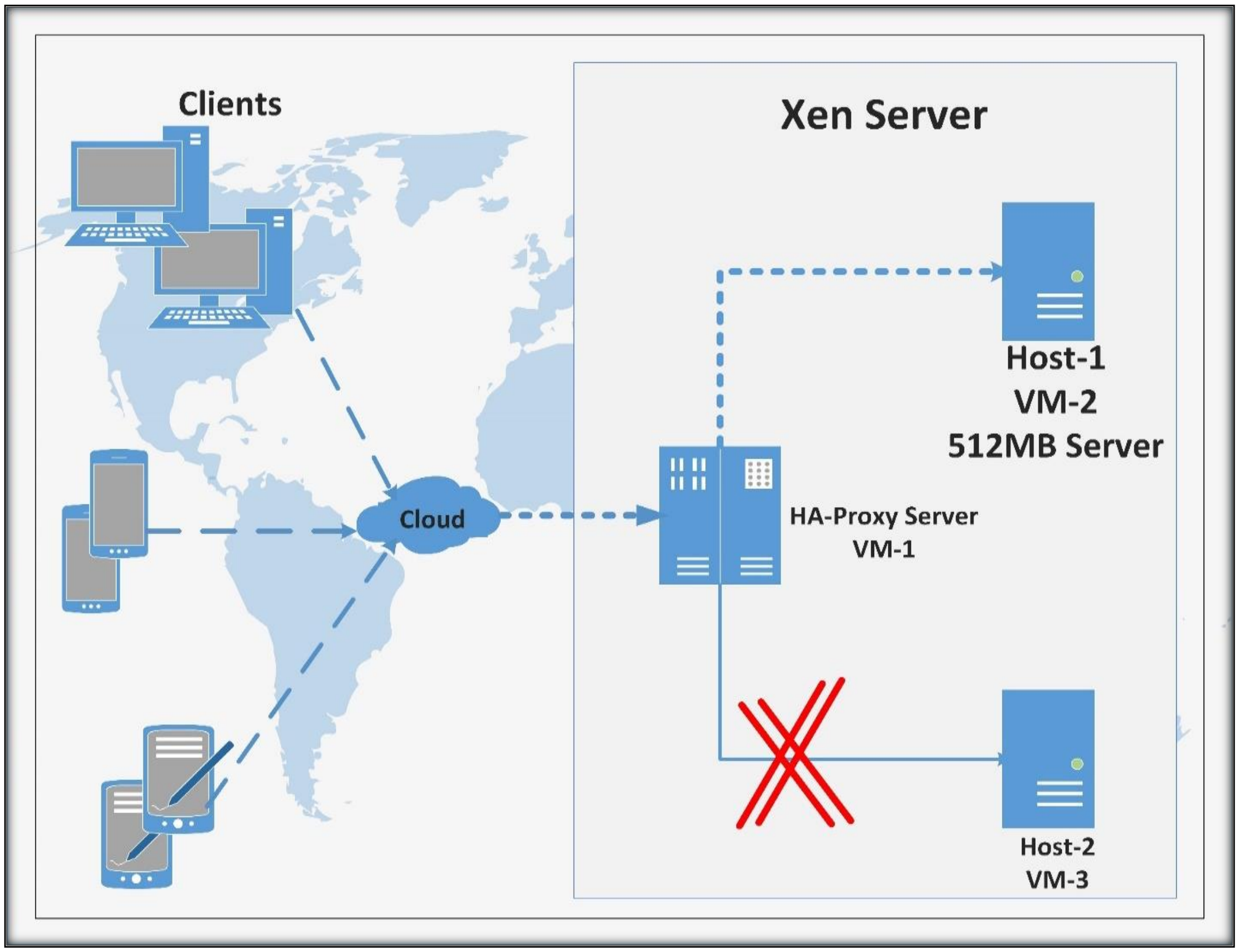

Figure 6: Single 512MB VM over Load Balancer

Curl request shows test script result where X-Forwarded is responsible to take client request and forward it to Server. Server IP is static as we have enable single server in the haproxy config.

```
shakeeb@Shakeeb-Laptop:~$ curl http://172.16.100.7/iptest.php
Backend Server IP: 172.16.100.5
Load Balancer IP: 172.16.100.7
Request Generated From: 172.16.100.4
shakeeb@Shakeeb-Laptop:~$ curl http://172.16.100.7/iptest.php
Backend Server IP: 172.16.100.5
Load Balancer IP: 172.16.100.7
Request Generated From: 172.16.100.4
shakeeb@Shakeeb-Laptop:~$ curl http://172.16.100.7/iptest.php
Backend Server IP: 172.16.100.5
Load Balancer IP: 172.16.100.7
Request Generated From: 172.16.100.4
shakeeb@Shakeeb-Laptop:~$ curl http://172.16.100.7/iptest.php
Backend Server IP: 172.16.100.5
Load Balancer IP: 172.16.100.7
Request Generated From: 172.16.100.4
```

After generating curl request to iptest file round we have analyzed that single host is active and no request is forwarded to Host-2. Now need to run apache bench test on one of sample web. Sample website is uploaded in the root path of the web directory. In this research sample web is uploaded in /var/www/html/ directory and update apache config file accordingly. Table.3 shows web stress test of 10th, 25th, 50th, 75th and 100th number of request. As per the observation response time increases so instantly on increasing number of concurrent request. Initial request response time is for 100 concurrent users is 206.4ms and reaches to 718.572ms after 100th request which is 3.5 times greater than its first request.

Table 3: Response time without Load balancing 512MB server

Percentage served	10users	20users	40users	60users	80users	100users
1	10.179	46.036	124.306	101.732	165.228	206.483
25	31.175	88.936	164.995	269.226	325.706	331.325
50	55.099	108.968	198.782	318.151	436.2	454.58
75	75.283	132.045	245.987	378.179	511.994	616.606
100	158.474	211.071	376.724	486.241	671.081	718.572

Figure.No. 7. Shows graphical representation when single 512MB webserver serves all client requests and the response time shows in milliseconds. Graphical results clearly represents as much as the number of concurrent request increases response time also increases while after increasing number of test, the load on the server also increases causes high load time.

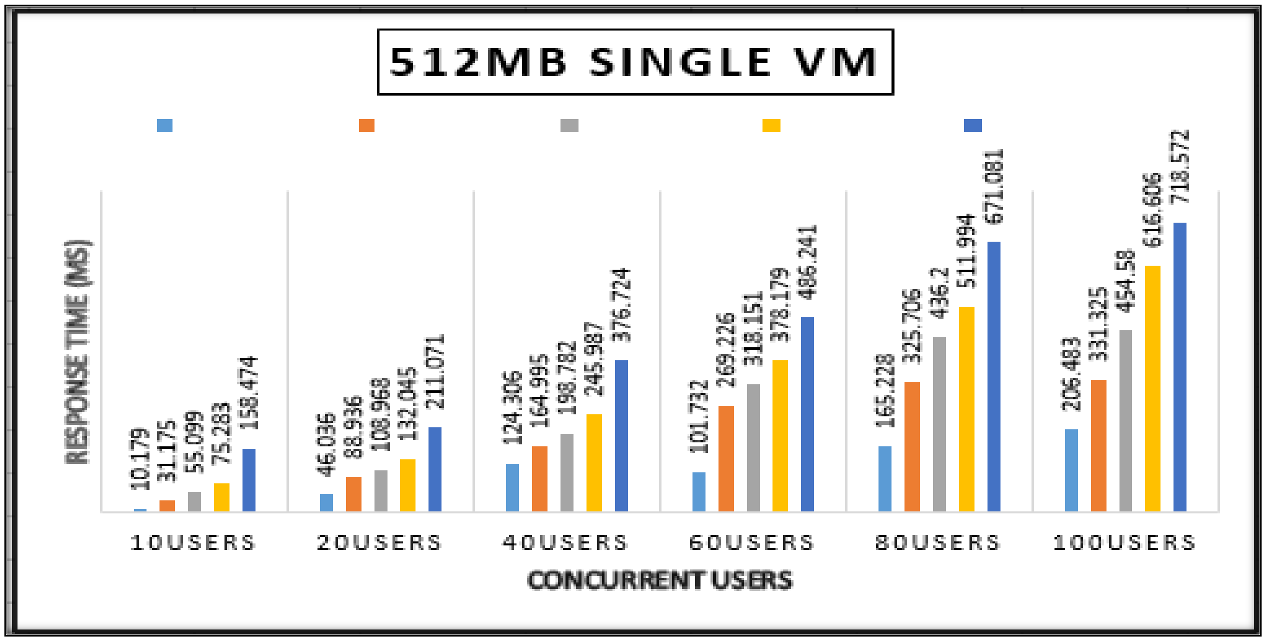

Figure 7: Response time without Load Balancing Server with 512MB

3.3.2. Web stress test over Host-2 with a memory to 1024MB

Second scenario has been tested by switching off all the web server except (Host-2) here host-2 has a memory of 1024MB and to analyze its behavior a curl request is generated. Also host-2 IP is activated on HA-Proxy Server so to deactivate round robin algorithm and all client request is handled by this only machine.

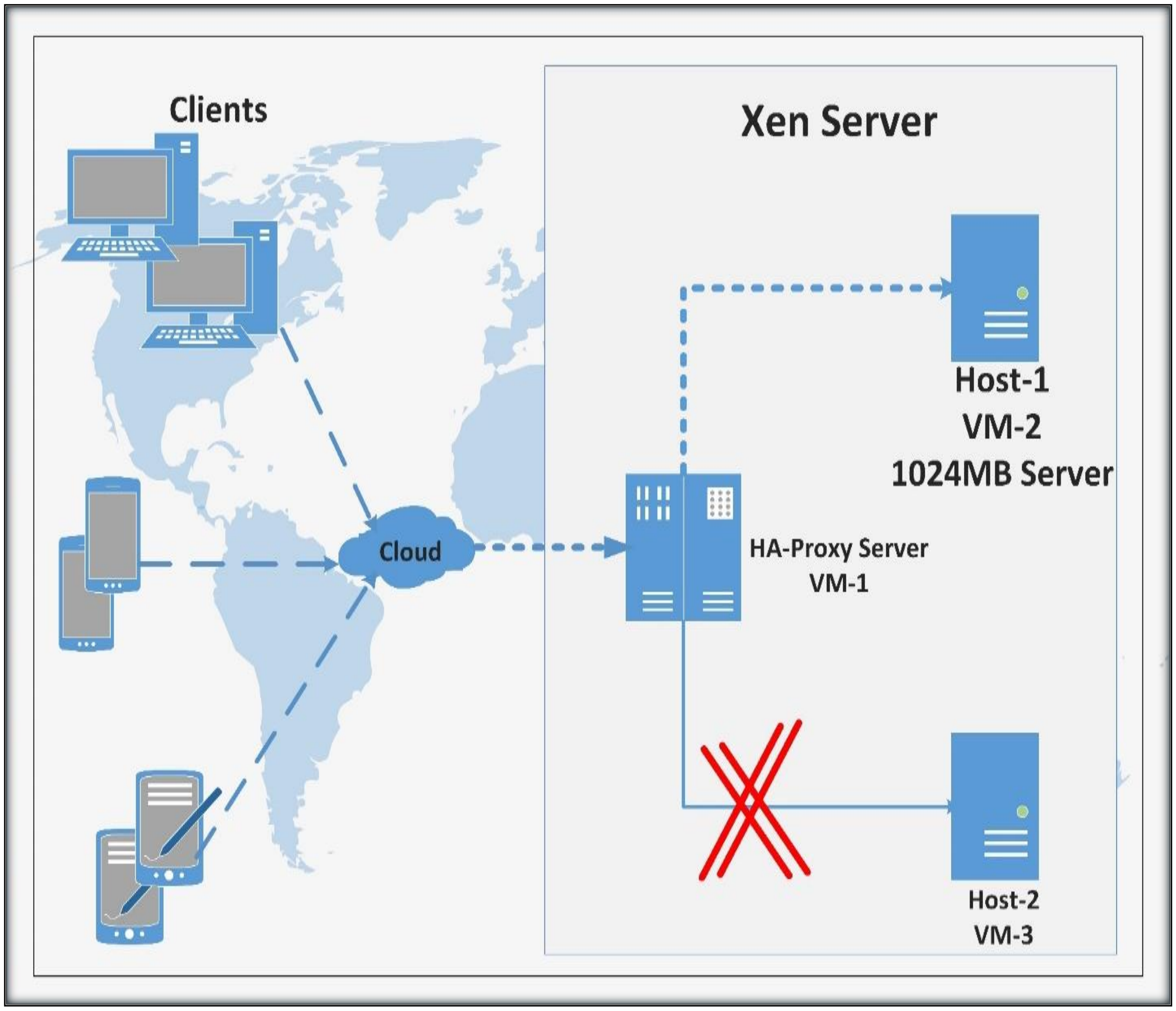

Figure 8: Single 1024MB VM over Load Balancer

This test is performed to compare it with third scenario that has equal memory. Server IP is static as we have enabled single server in ha-proxy configuration.

```
shakeeb@Shakeeb-Laptop:~$ curl http://172.16.100.7/iptest.php
Backend Server IP: 172.16.100.6
Load Balancer IP: 172.16.100.7
Request Generated From: 172.16.100.4
shakeeb@Shakeeb-Laptop:~$ curl http://172.16.100.7/iptest.php
Backend Server IP: 172.16.100.6
Load Balancer IP: 172.16.100.7
Request Generated From: 172.16.100.4
shakeeb@Shakeeb-Laptop:~$ curl http://172.16.100.7/iptest.php
Backend Server IP: 172.16.100.6
Load Balancer IP: 172.16.100.7
Request Generated From: 172.16.100.4
shakeeb@Shakeeb-Laptop:~$ curl http://172.16.100.7/iptest.php
Backend Server IP: 172.16.100.6
Load Balancer IP: 172.16.100.7
Request Generated From: 172.16.100.4
```

By analyzing above it only shows single server is activated. After generating apache stress test over load balancer server with single activated Virtual machine having memory 1024MB. Table.4 shows web stress test of 10th, 25th, 50th, 75th and 100th for specified number of concurrent requests. As per the observation response time increases so instantly on increasing number of concurrent request. Initial request response time for 100 concurrent users is 195.4ms and reaches to 581.217ms after 100th request which is 3 times greater than its first request and after comparing it with low memory host it response time is much better.

Table 4: Response time without Load balancing 1024MB server

Percentage served	10users	20users	40users	60users	80users	100users
1	12.787	68.728	107.622	132.802	170.413	195.495
25	42.152	94.263	189.304	270.557	289.051	316.663
50	61.389	117.537	232.709	321.436	405.082	447.792
75	74.126	137.045	261.173	363.685	474.788	581.217
100	110.572	193.713	358.619	462.495	570.491	666.255

Below image shows graphical representation of the response time of 1024MB webserver. We clearly analyze the response time of 1024MB webserver is less as compare to 512MB webserver. If we increases system memory response time to cater client requests become low and helps in fast browsing of data.

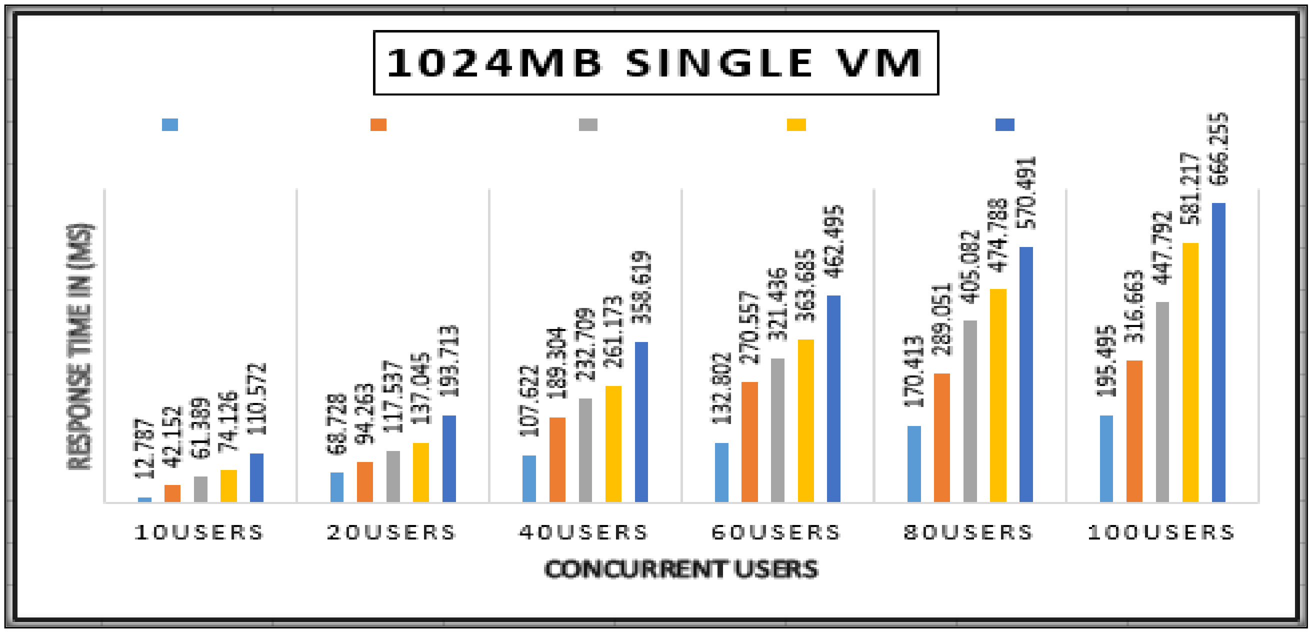

Figure 9: Response time without Load Balancing Server with 1024MB

3.3.3. Web stress test over load balancer using two 512MB webservers.

Third apache bench test is performed on two 512MB webserver which is used as a load balancer also presented in Figure. No. 10 .Minimum request time for single user is approximate same as previous scenarios but as soon as the number of concurrent request increases client requests handled efficiently and response time comes relatively low as compare to single 512MB and 1024MB web servers.

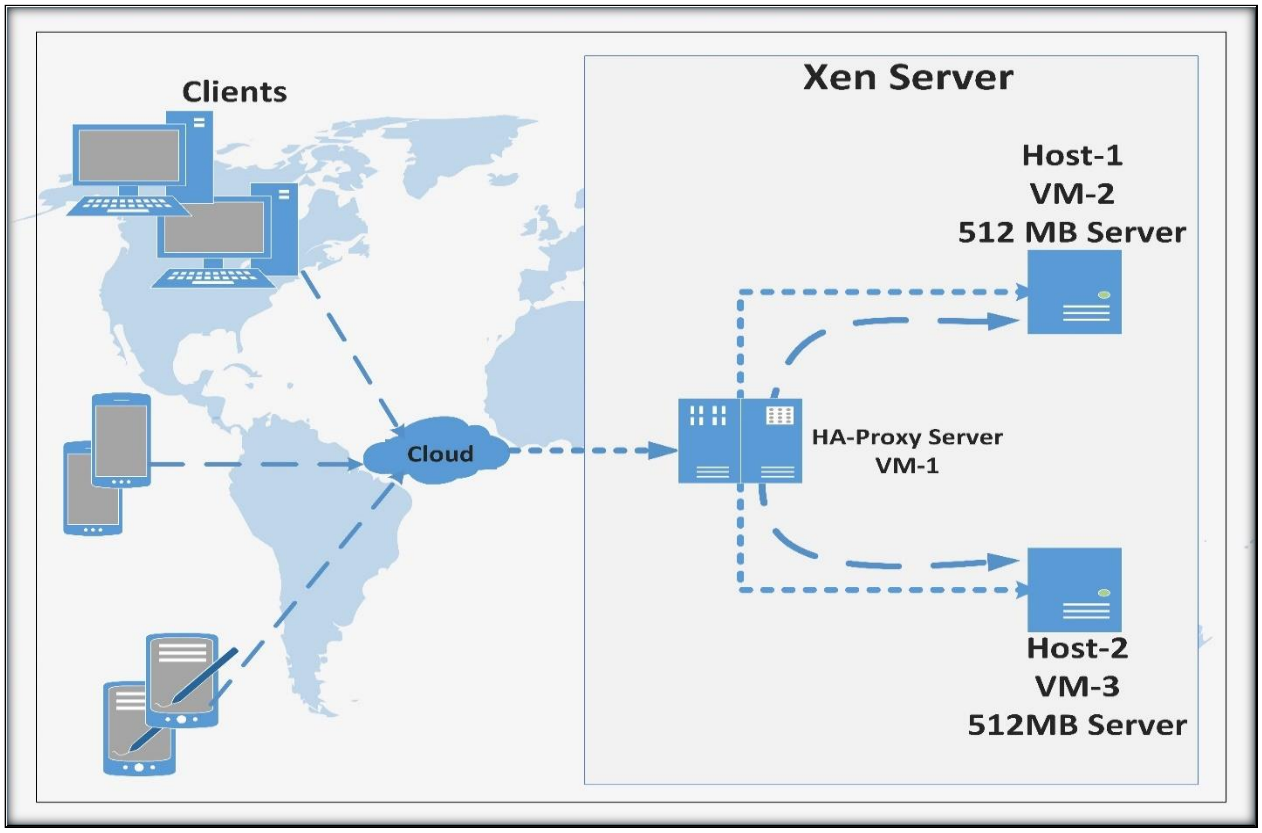

Figure 10: Load Balancer using round robin algorithm over VM

Below Table.5, shows the result of 10th, 25th, 50th, 75th and 100th request result with defined number of concurrent users over load balancer. As per experimental analysis if the number of specific request increases the response time to surf single request also increases but have far better response time. When a first requests generated response time is little high as first request is handled by HA-Proxy and then it forwards the request to backend defined servers. After 100 concurrent users first request takes 241.282ms and reaches to 481.57ms which is approx. 2 times of initial request which clearly shows that it reduces with a value of 185ms after using load balancer.

Table 5: Response time over Load balancer

Percentage served	10users	20users	40users	60users	80users	100users
1	21.007	31.954	97.309	137.762	172.273	214.282
25	35.029	71.374	119.66	181.67	249.96	266.66
50	38.72	90.407	166.45	229.07	262.16	349.49
75	48.636	103.658	191.02	279.3	331.55	399.52
100	78.748	148.37	239.49	359.45	439.24	481.57

For all apache bench test over HA-Proxy server after enabling load balancer presented in Figure No. 11 From all presented experiments response time using load balancer server is far better as compare with using single webserver with same system configuration.

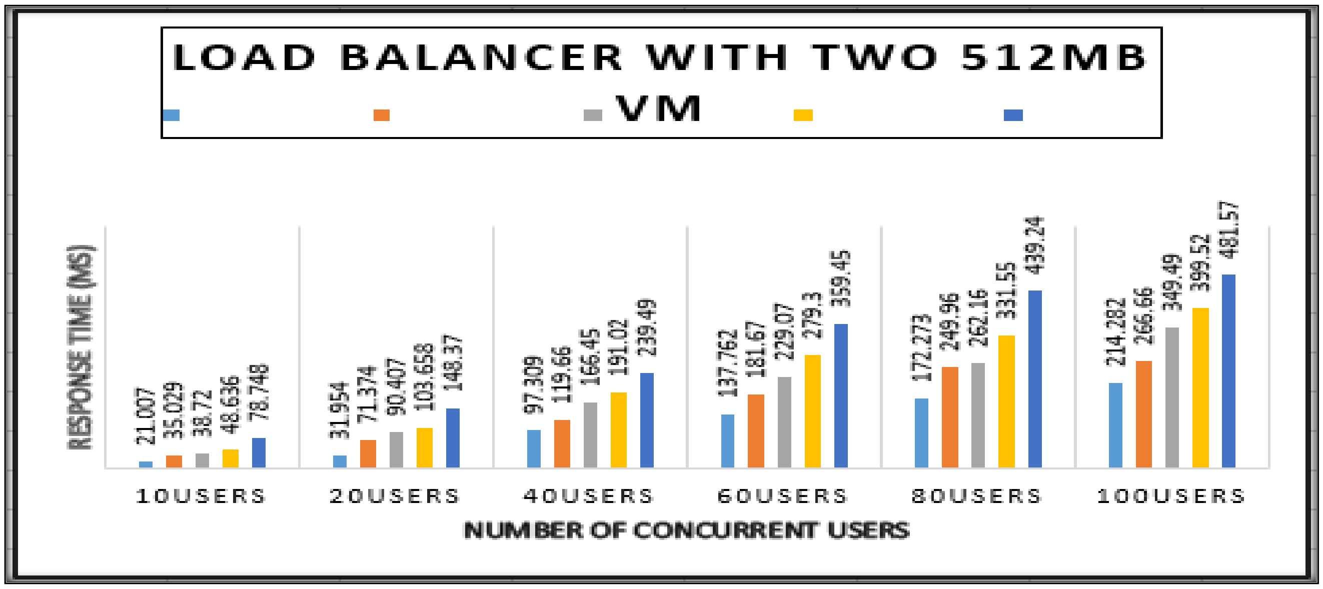

Figure 11: Response time with Load Balancing Server

3.4. Virtual Machine Experimental Comparison:

Experiments performed on these virtual machines helps to analyze the behavior and investigate the result after 100 concurrent request shows that load balancer response time is far better as compare to other two experiments either using 512MB or using high configuration virtual machine.

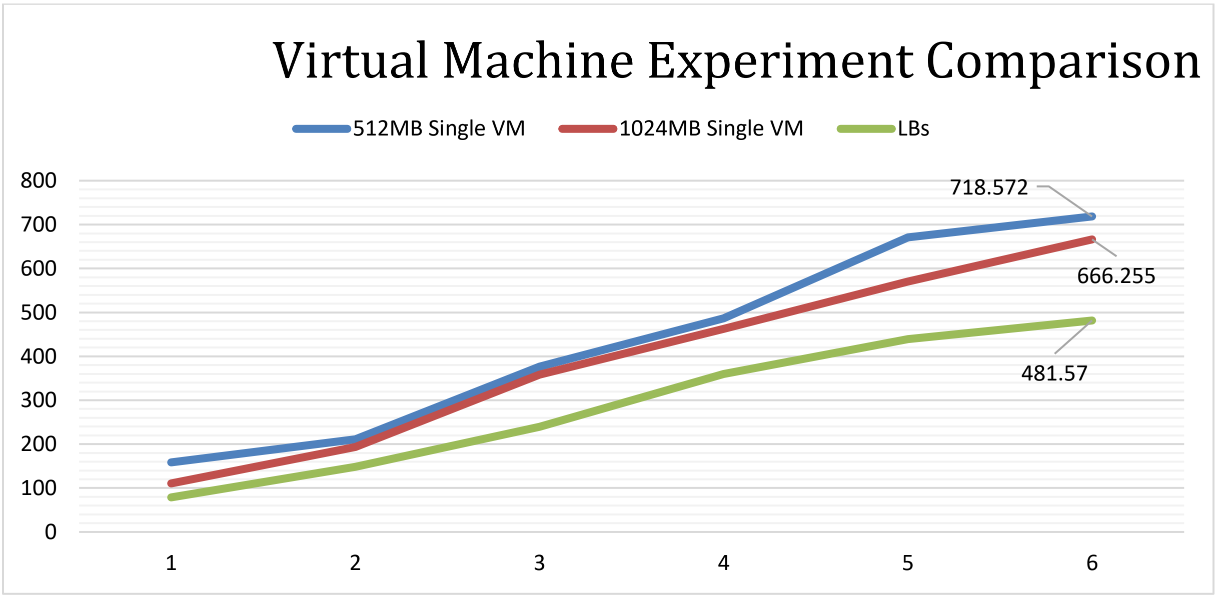

Figure 12: VM Experiment Comparison

3.5. CONCLUSION:

Experiments performed on those virtual machines helps to analyze the behavior and investigate best possible method used between load balancer with HA-proxy and single virtual machine.. It can also be concluded from the experiments that software switching is more reliable and efficient as compare to hardware switching in real time. This virtualization scheme improves the utilization of hardware which reduces the cost and its maintenance.

CHAPTER 4: EXPERIMENTAL ANALYSIS OF LOAD BALANCER WEBSERVERS OVER DOCKERIZATION

Container a Docker based virtualization has recently emerged and known for its lightweight technology and gain enough popularity in the community of software development [12], [13]. These light weight containers are build, test and move with their dependencies to different distros and location. Starting these container is much fast as compare to virtualization [13], [14]. Dockerization challenges system resource, speed and performance with respect to virtualization for system development process and has great advantage because of providing efficient workflow to the developer [14], [15], [16]. Therefore, Docker a container service platform empowers system admin and developers to build, ship and run these applications anytime anywhere [17] therefore also known as Containers-as-a-Service (CaaS).

Docker an open source container virtualization platform runs separately from host infrastructure and helps in running, managing and developing several applications. Its aim is to ship code faster and deploying it to any of container based technology [36] .Due to the isolation of container from infra it allows to run many containers on a single host simultaneously without interrupting other containers. We can limit these containers constrained to use specified amount of available system resources as they share same kernel and for better efficiency limiting those containers is possible.

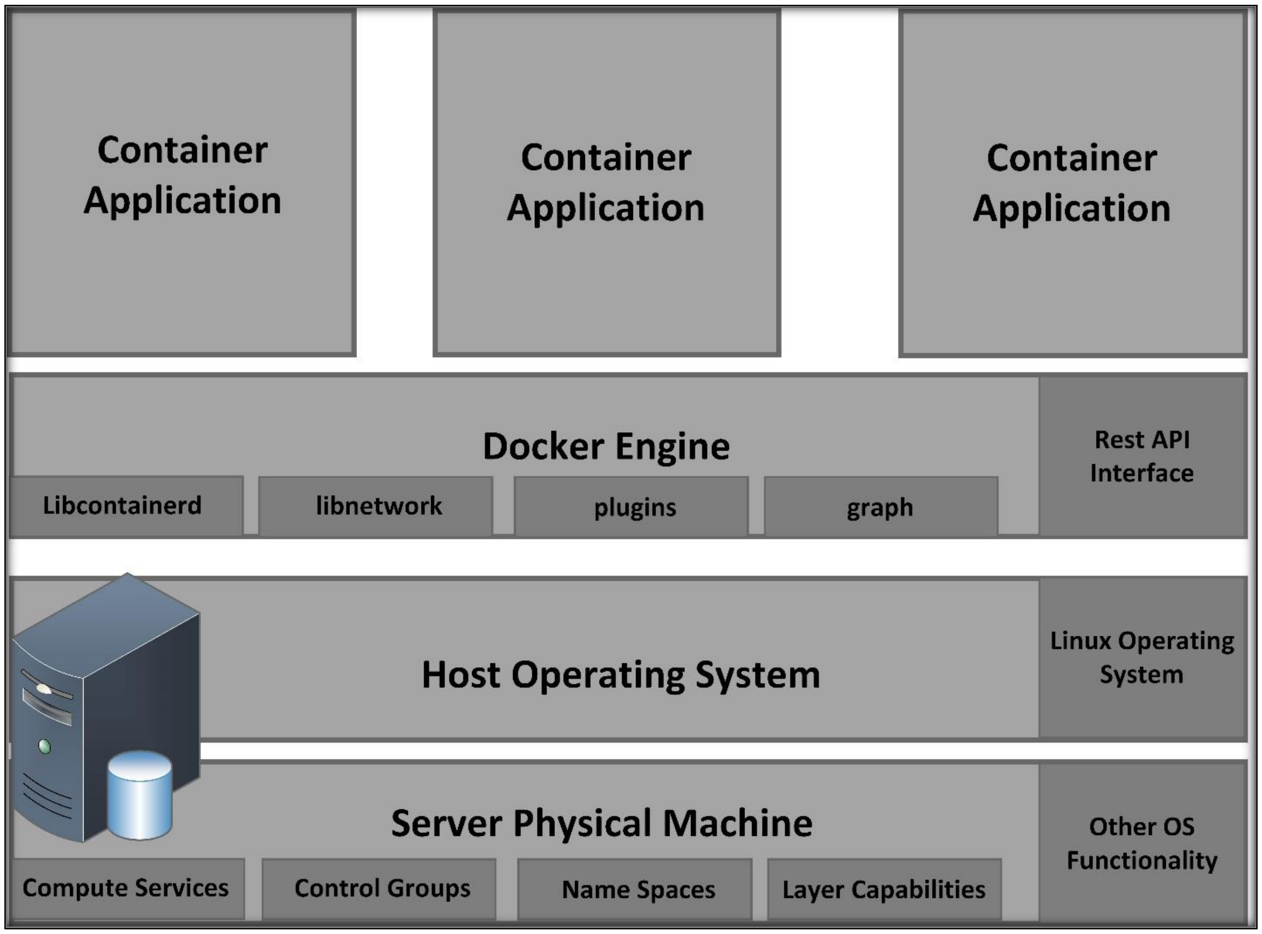

Figure 13: Docker Container Architecture

Container is relying on Kernel's functionality therefore it doesn't requires separate operating system as compare to VM and These containers uses separate name space to completely isolate the application's view of the OS. It gives benefit to container for rapid deployment with performance in memory, disk, central processing unit and its network. Docker implements high-level application program interface which is responsible to run containers with their unique process ID space, network space and structure of its file system. Hardware utilization of these containers is very high as they run without extra load of a hypervisor.

4.1. System Design

System configuration may vary according to the demand and utilization of the application services. Docker uses server-client architecture, Client containers communicates with docker daemon presented in the host machine. Both the docker container and docker daemon is placed in a single machine. Docker daemon is responsible for running, building, distributing the resources between the containers. Docker daemon can communicate with containers through a bridge called docker0. Docker consist of three major components: Docker registries, Docker container and Docker images.

4.1.1. Docker Registries

Docker registry contains built-in images which is private or public store of Docker images. Docker registries is the place from where user can download and upload designed images.

4.1.2. Docker Containers

Docker container holds everything needed to run an application. It looks like an OS containing OS image / file system / system packages. These containers can be created from existing Docker images and can be start, stop, move, run and delete.

4.1.3. Docker Image

Docker image is designed template used to create Docker containers. For example: Image contains a debian8.0 OS with php5.6 and apache2 and other packages like vim an editor, net tools for network configuration using command line. These images can be created with desired requirements and can be re-use to create new containers.

To create an image DockerFile is required which is a text document which contains, number of instructions to build container with our desired packages.To create a docker container image you need to store the file in any folder and runs below command. Docker file helps to create same image where ever we need that image. Below is the Dockerfile content which pulls all data from docker hub.

```
root@docker:/home/shakeeb/thesiswork# docker build .
```

```
root@docker:/home/shakeeb/thesiswork# cat Dockerfile
FROM debian:jessie
FROM php:5-apache
WORKDIR /var/www/html
EXPOSE 80
```

Here it represents that the image is created with a debian OS and install php and apache. PHP is required for the development use and required for web design. After that default working directory /var/www/html is set and expose command helps to tell the Docker that it listens on port 80 which is the default port to access web pages. By default expose assumes that it is TCP connection. In this research Linux OS is installed on single physical machine and on top of Docker engine. Two container images were created using Docker file. Each container image consist of same data set, system packages and utilities. We have used these images with our different experiments.

Table 6: Host Machine Configuration Table

System Conf.	Host OS	Container-A	Container-B
Architecture	x86_64	default	Default
OS	Debian 3.16.513+deb8u1	Debian3.1	Debian3.1
Processor	2712.002 MHz	Default	Default
RAM	1024MB	512MB	1024MB
NIC	1	Virtual Interface	Virtual Interface
Disk	10GB	Default	Default

4.2. Load Balancing Using HA-Proxy on Containers

Round robin is the most common one supported by major cloud providers is the common most available algorithm [38]. In this paper two apache webserver based images is created with same packages. Static IPs assign to these containers with a CLI, Container-A with 172.16.200.15 and Container-B with 172.16.200.20. Table No.7 shows Container ID which is automatically and uniquely assigned while creating container from image.

Table 7: Docker Stats Container Table

CONTAINER ID	CONTAINER NAME	MEM USAGE / LIMIT	IP
64cf27654ac7	Container-A	1.066MiB / 489MiB	172.16.100.15
eefc06227ab6	Container-B	976KiB / 1000MiB	172.16.100.20

HA-proxy is installed on Linux host machine. All client request is handled by this machine and sent traffic to Docker Network Bridge which is responsible to transfer traffic using round robin algorithm to defined containers in its pool to test the load balancer iptest.php code is written which shows load balancer result containing web server IP, Load Balancer IP (HA proxy server) and X-Forwarded IP (A system which request for this page).

To analyze round robin algorithm curl request is generated contains same code presented in previous chapter i.e. chapter 3. The code will pull the load balancer server ip, the client from which request is generated and backend server which handled the client request

```
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.15
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.20
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.15
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.20
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
```

Each client request has been verified by seeing that round robin algorithm works without any data loss and request has been handled by each container consecutively. First request is catered by the Container-A and second request by Container-B, 3rd request is surfed by Container-A again and fourth by Container-B and so on which clearly represents behavior of HA-Proxy defined round robin algorithm. We are using only two web containers so only two IPs appears as a server IP.

4.3. Self-Organizing of Web Container before Service Degradation.

In this paper two web containers were created to cater client request and if there is any service degradation than subsequent hosts is required to handle further request. To predict high load we have developed a script which checks system load and on certain defined threshold another container is created with predefine image which contains same data set and system packages. Now a day’s people create scripts to define program for all UNIX environment to break all complex projects into simple smaller task that is also known as bash scripting (Bourne again Shell) in Linux environment. In this paper an algorithm is developed using a shell scripting which uses a way to explore the capabilities in Linux environment [39] using same algorithm we have designed customized algorithm which checks all the updates from primary server. To examine system performance, load and memory test we are using a Perl script and on the basis of it if there is high load a container is automatically created with a static IP which is automatically add in HA-Proxy conf by our designed algorithm. This script is placed inside main server which is responsible for to analyze the system load/memory. Using our designed algorithm the container is created with in a second or two and rest of client request is handled by this newly added container.

The algorithm is written in bash scripting with single defined local IP so that it created on same host and work as a 3rd node to handle user request. Static IP assign to the third host shown in script is 172.16.100.25 and the container image name is container-b-512mb-clone. So it clearly represents that already existed image is used to create third container.

```
#!/bin/bash
a=$(/check_memory -f -c 2 | awk -F " " {'print $1'})

echo $a

############## Checks Condition if Status Critical than Runs below ##########

if [[ $a == CRITICAL ]];

then

docker  run  -itd  --memory=489M  --memory-swap=509M    --net  ipstatic  --ip
172.16.200.25 container-b-512mb-clone

containername=$(docker ps -a | awk -F " " {'print $1'} | head -2 | tail -1)

echo  "New  Container  with  ID:  $containername  created  on  `date  '+%Y-%m-%d
%H:%M:%S'`"

docker ps -a > viewdocker

docker exec -it $containername /etc/init.d/apache2 start

cat viewdocker


################ Now edit HA-PROXY Conf File with New IP #################


echo "    server lamp3 172.16.200.25:80 check" >> /etc/haproxy/haproxy.cfg

/etc/init.d/haproxy reload

echo "Container added in HA-Proxy on `date '+%Y-%m-%d %H:%M:%S'` and is active
on Load Balancer"

################# if Status OK than ignore above and runs below ##############

        else
        echo "every thing is fine"

        fi

#################################    End         ############################
```

The output is shown below once new container created after high load.

```
OK
571bfc8d29e72d506ffe413c5b88e847e6a3e360014cd8e839e9bdad65e3bc5b
New Container with ID: 571bfc8d29e7 created on 2018-07-23 07:26:37
[....]  Starting  web  server:  apache2AH00558:  apache2:  Could  not  reliably
determine the server's fully qualified domain name, using 172.16.200.25. Set
the 'ServerName' directive globally to suppress this message
. ok
CONTAINER ID        IMAGE                    COMMAND                  CREATED
STATUS                  PORTS                NAMES
571bfc8d29e7         container-b-512mb-clone    "docker-php-entrypoi…"    Less
than a second ago   Up Less than a second   80/tcp            priceless_feynman
eefc06227ab6       apacheweb                "docker-php-entrypoi…"   4 months
ago               Up 21 minutes           80/tcp              Container-B
64cf27654ac7       apacheweb                "docker-php-entrypoi…"   4 months
ago               Up 15 minutes           80/tcp              Container-A
[ ok ] Reloading haproxy configuration (via systemctl): haproxy.service.
Container  added  in  HA-Proxy  on  2018-07-23  07:26:38  and  is  active  on  Load
Balancer
```

571bfc8d29e7 is newly created container and as seems that 80/tcp is also become active just after this container creation. Below is the response for three consecutive request which clearly shows that first request handle by Container-A, 2nd by Container-B and the third by newly created container.

```
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.15
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.20
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.25
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.15
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
```

Perl based plugin code is presented below which is responsible to pull three stats of memory from all Linux OS (including FreeBSD / Debian / Ubuntu / CentOS etc.) .

a) OK if current memory is in our predefined limit.

b) CRITICAL if current memory reaches to defined limit in script.

c) Warning is like a predictable value (Not using this in this research)

The plugin generates critical status if 2% memory of the system is left. The script doesn't contain the exact path here as we have run it with our prototype machines which gives response with in few milliseconds and if there is high memory than create a container with in few seconds. Below is the flow chart represents how the scripts run and when it get triggered to create a container.

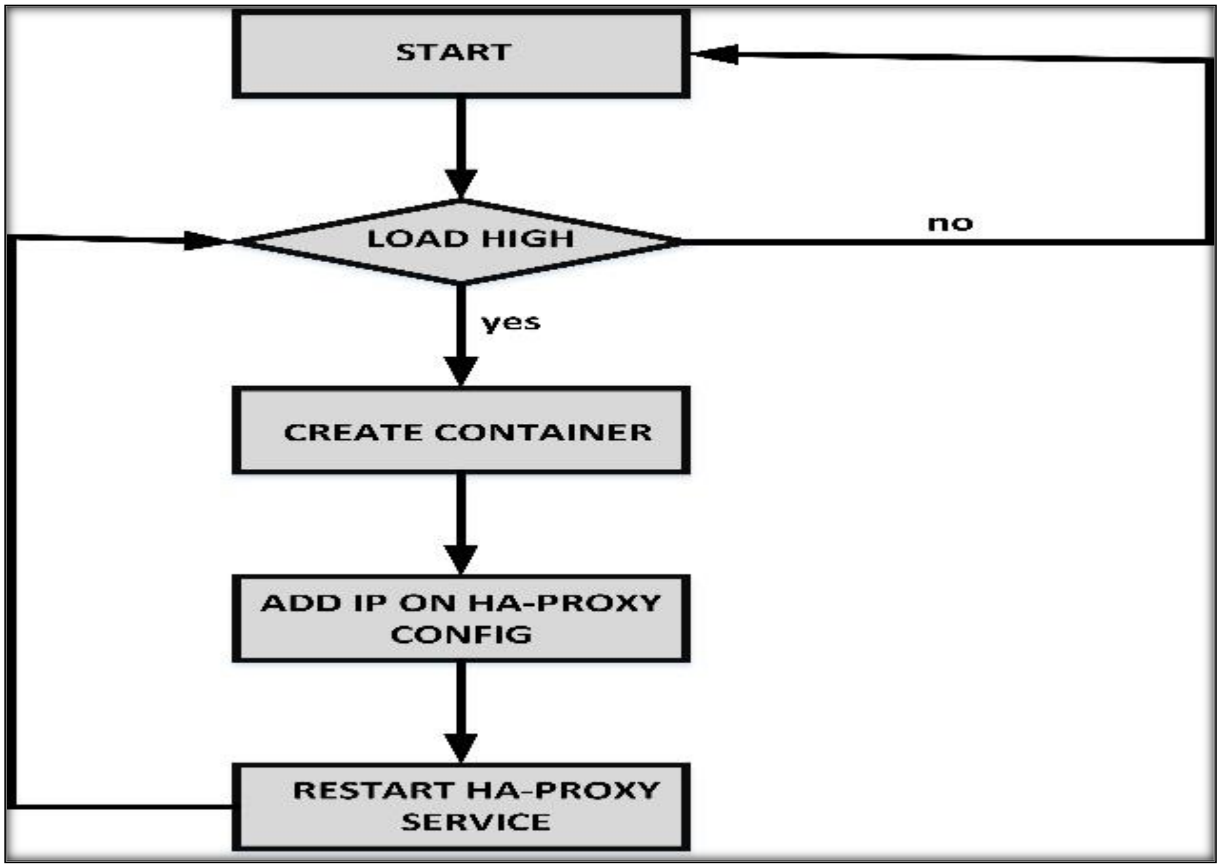

Figure 14: Flow Chart of Self-organized web container

4.4. Experimental Analysis of Load Balancer

To evaluate the behavior and response time of the container while using same data set in this research list of below experiments performed that measures response time of containers. Each response time is pull from same apache bench test result and stored it on excel sheet to analyze the difference.

These experiments performed

a) Apache bench stress test using single web container (512MB Container).

b) Apache bench stress test without using multiple webservers (Single 1024MB Container).

c) Web stress test over load balancer using two 512MB web containers.

All experiments have been performed to check the performance on HA-Proxy server by increasing number of concurrent connection from 1user to 100users and then analyze response time for each request. These results based on number of connections, response time, connection rate, and timeouts. We are using a same appsea demo webpage with a size of 1,130,489 bytes so that to analyze same page result. In this research Class-B IP pool is used for containers. HA-Proxy machine static IP set to 172.16.200.1 which is responsible to forward all client request to back end containers. Below is the curl command run to evaluate active host status, load balancer IP and backend container.

```
Curl http://172.16.200.1/iptest.php
```

After that apache bench test is performed on host using command below.

```
shakeeb@Shakeeb-Laptop:/mnt/c/Users/goto$  ab  -e  10users-2  -n  100  -c  10
http://172.16.200.1/appsea
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 172.16.200.1 (be patient).....done

Server Software:        Apache/2.4.10
Server Hostname:        172.16.200.1

Server Port:            80

Document Path:          /appsea
Document Length:        313 bytes

Concurrency Level:      10
Time taken for tests:   0.269 seconds
Complete requests:      100
Failed requests:        0
Non-2xx responses:      100
Total transferred:      54000 bytes
HTML transferred:       31300 bytes
Requests per second:    372.27 [#/sec] (mean)
Time per request:       26.863 [ms] (mean)
Time per request:       2.686 [ms] (mean, across all concurrent requests)
Transfer rate:          196.31 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   0.5      1       3
Processing:     8   25   5.6     25      41
Waiting:        7   22   5.8     22      41
Total:          9   26   5.6     26      43

Percentage of the requests served within a certain time (ms)
  50%     26
  66%     28
  75%     29
  80%     30
  90%     32
  95%     34
  98%     39
  99%     43
 100%     43 (longest request)
```

4.4.1. First Scenario

It has been tested by switching off all the web containers except (Container-A) has a memory of 512MB shown in tables No.7 and to analyze its behavior a curl request is generated. Curl request shows test script result where X-Forwarded is responsible to take client request and forward it to Server. Here Server IP is static as we have enabled single server in HA proxy so it shows IP of Container-A.

```
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.15
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.15
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.15
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.15
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
```

Table No.8 shows that when 100 concurrent users generate request the container response time after 100th requests reaches to 241.577ms with a ratio of 5 with respect to initial request.

Table 8: Response time single 512MB Container

Percentage served (%)	10users (ms)	20users (ms)	40users (ms)	60users (ms)	80users (ms)	100users (ms)
1	11.411	12.181	25.803	34.362	42.435	47.354
25	24.075	42.61	76.85	98.318	106.653	107.995
0	26.312	48.49	95.581	125.662	147.939	157.279
75	29.284	57.302	106.554	142.2	174.401	200.47
100	35.358	74.791	119.412	176.641	204.228	241.577

Graphical representation is presented in Figure. No. 14 when single 512MB web container serves all client requests. As soon as the concurrent request increases response time also increases.

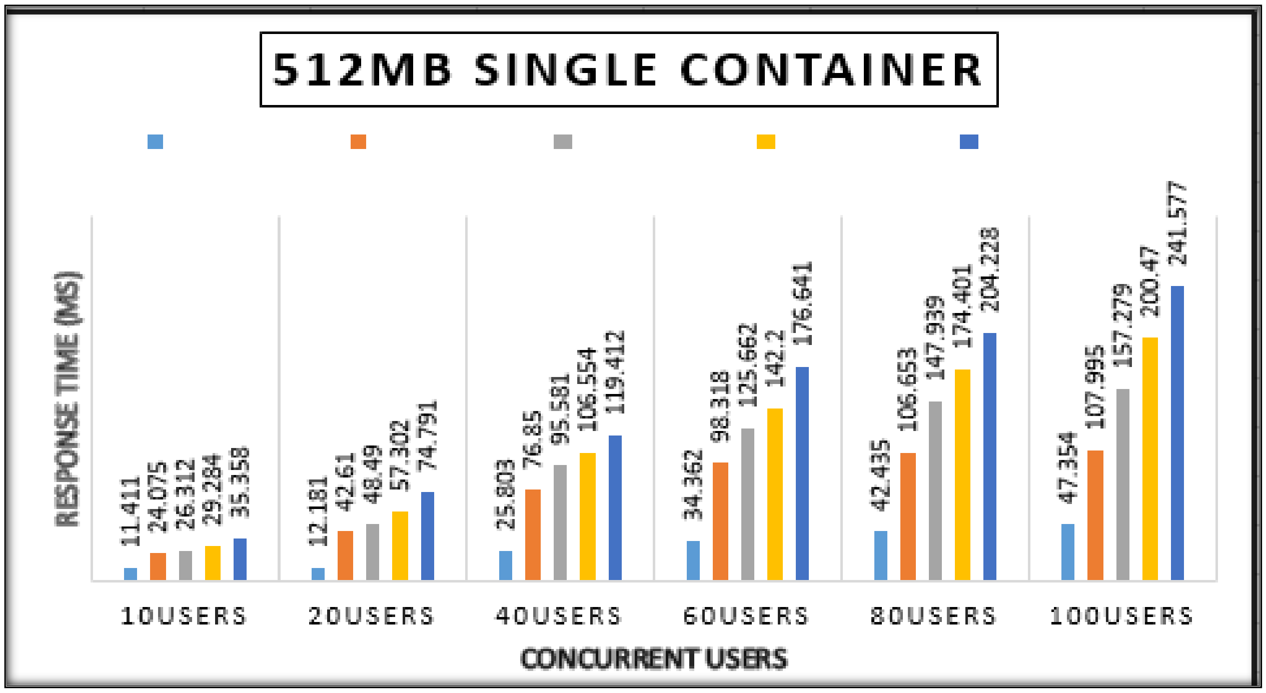

Figure 15: Response time with 512MB Single Container

4.4.2. Second Scenario

It has been performed over 1024MB memory single container (Container-B) which is equivalent to the double of total memory of previous container. To analyze the behavior a curl request is generated over same data set container.

```
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.20
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.20
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.20
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
shakeeb@Shakeeb-Laptop:~$curl http://172.16.200.1/iptest.php
Backend Server IP: 172.16.200.20
Load Balancer IP: 172.16.200.1
Request Generated From: 172.16.100.1
```

By analyzing above it only shows single container is active. After generating apache stress test over load balancer server with single activated 1024MB memory container. Table No.9 shows response time of generated requests with respect to concurrent users. Here Initial request response time is less as compare to 512MB container which clearly shows that higher memory response is far better as compare to low memory containers.

Table 9: Response time single 1024MB container

Percentage served (%)	10users (ms)	20users (ms)	40users (ms)	60users (ms)	80users (ms)	100users (ms)
1	10.537	16.33	24.915	33.01	41.507	52.025
25	21.111	45.184	68.248	85.985	98.736	95.908
50	24.64	51.816	93.826	112.517	137.563	133.154
75	27.097	57.45	106.349	128.741	154.654	171.068
100	33.392	77.459	126.68	159.641	185.452	206.025

Graphical representation is presented in Figure. No. 15 when single 1024MB web container serve all client requests and when concurrent request with number of test increases response time also increases.

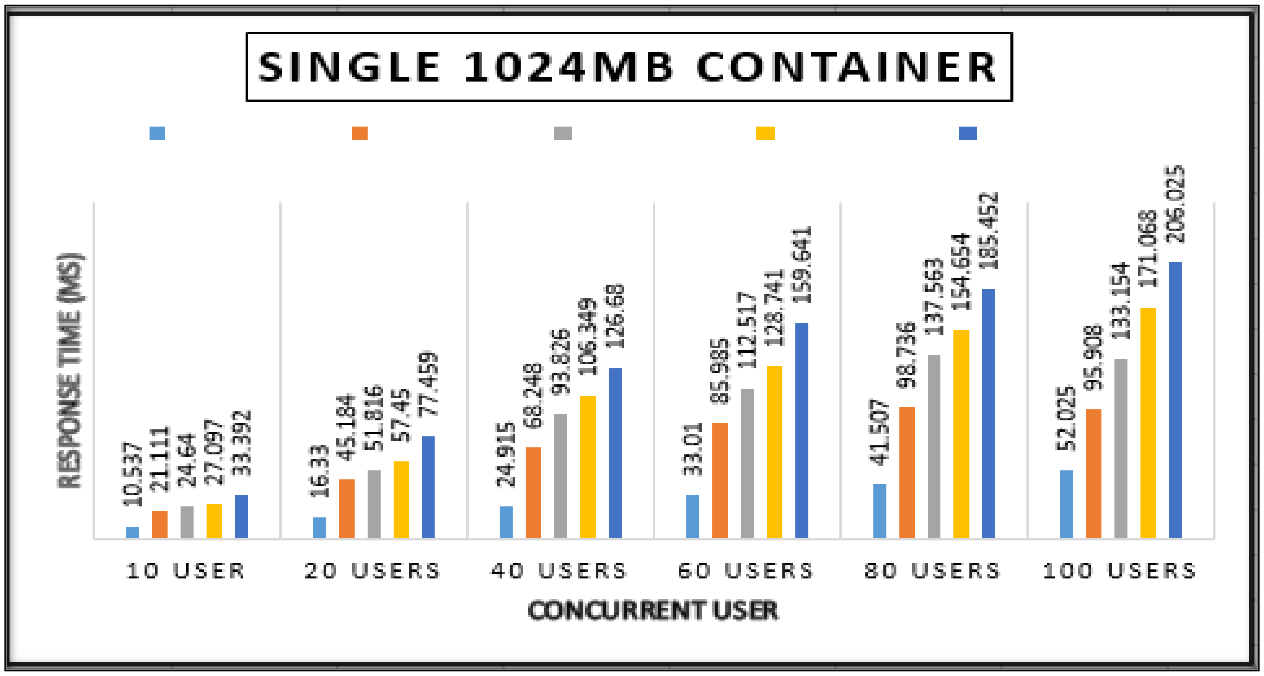

Figure 16: Response time with single 1024MB container

4.4.3. Third experiment

It has been performed on two 512MB web container hosted on single machine which is used as load balancer. In Figure No. 16 it clearly presents all client request is received by the main host and on backend containers were present which handles client request. Container-A and Container-B here has same memory.

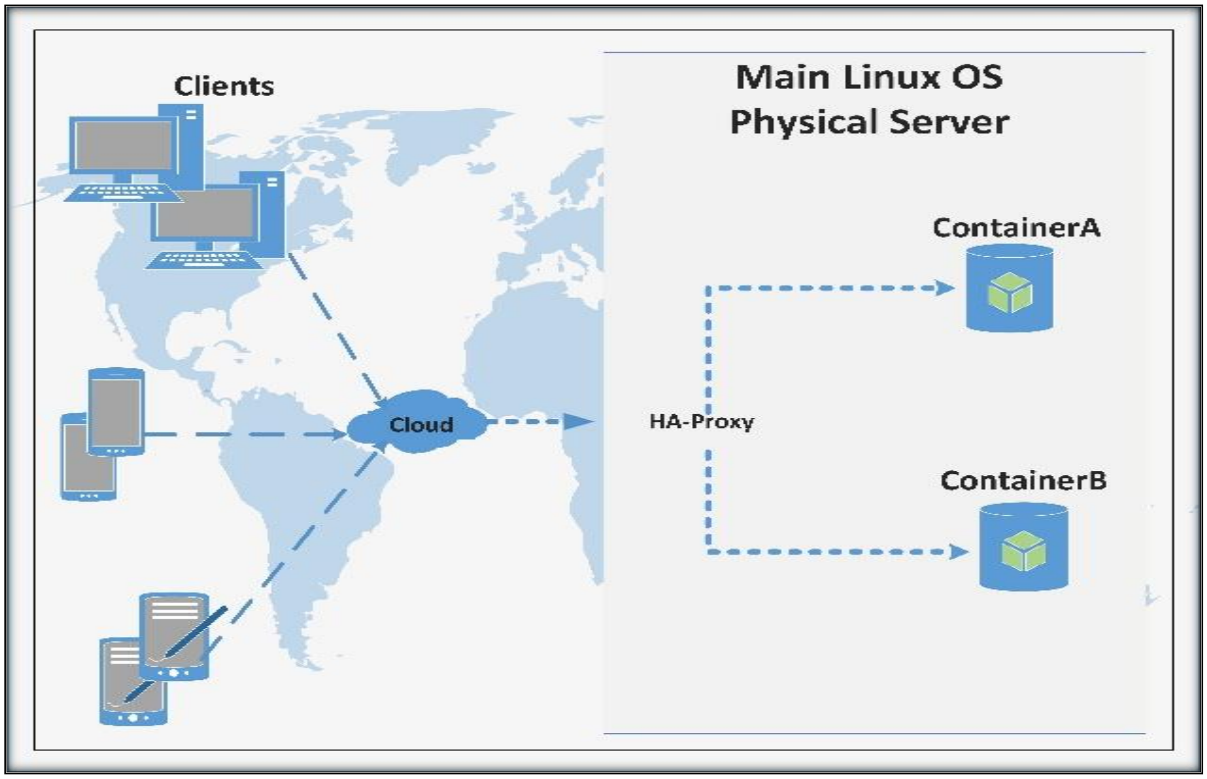

Figure 17: HA-Proxy Architecture for container

After sending client request to main HA-Proxy server minimum request time for 10user after 100requests is approximate same as previous scenarios but as soon as the number of concurrent request increases its response time becomes far better than 1024MB server.

Table 10: Response time with Load Balancing Containers

Percentage served (%)	10users (ms)	20users (ms)	40users (ms)	60users (ms)	80users (ms)	100users (ms)
1	8.804	19.291	28.155	37.878	32.272	51.488
25	21.273	40.702	71.214	87.939	95.012	96.241
50	23.44	49.148	93.505	113.945	133.14	133.162
75	27.096	54.137	105.709	138.75	153.966	166.305
100	32.243	71.862	128.666	160.008	194.475	201.757

Graphical representation is presented in Figure. No. 17 when two web container serves client requests in a round robin fashion. Initial request takes some time as it need to decide to which server the request need to be forward.

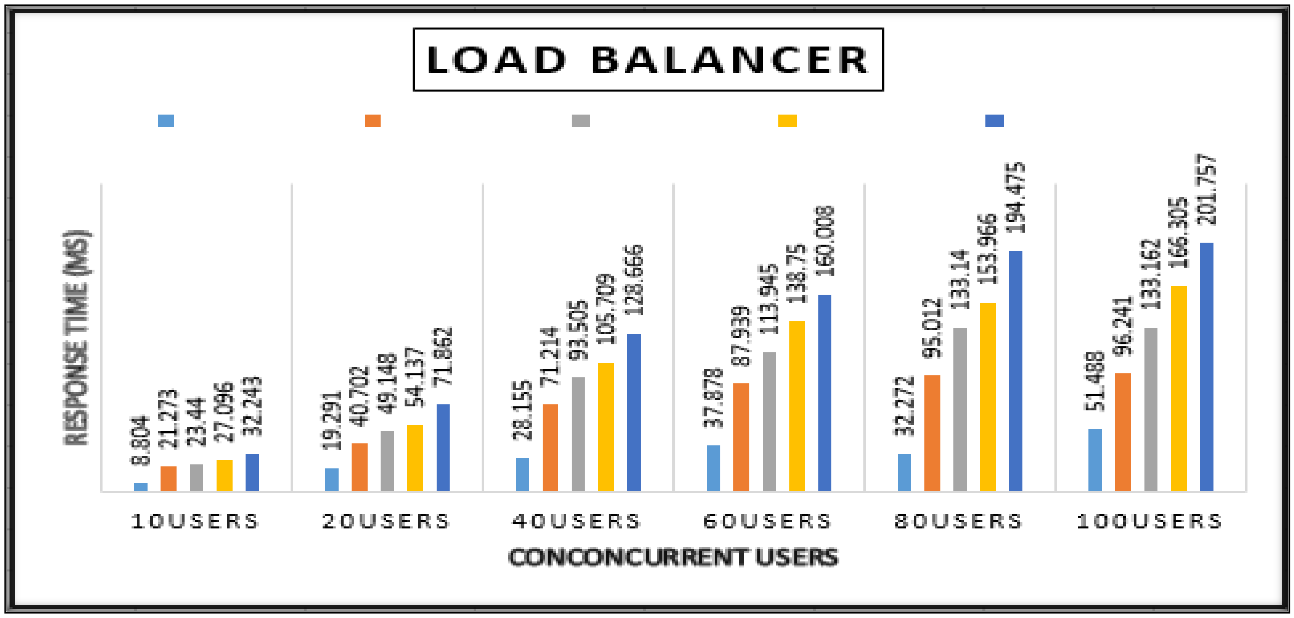

Figure 18: Response time of Load Balancing Containers

For all apache bench test over HA-Proxy load balancer server presented in Figure. No 17. Shows that after 100 concurrent request response time is 201ms which is lesser than 241ms (Experiment 1).

4.5. Performed Experimental Analysis of Containers.

These experiments clearly represents that using load balancer over container gives far better response time as compare to using single container. Below graph clearly shows the final response after 100 requests.

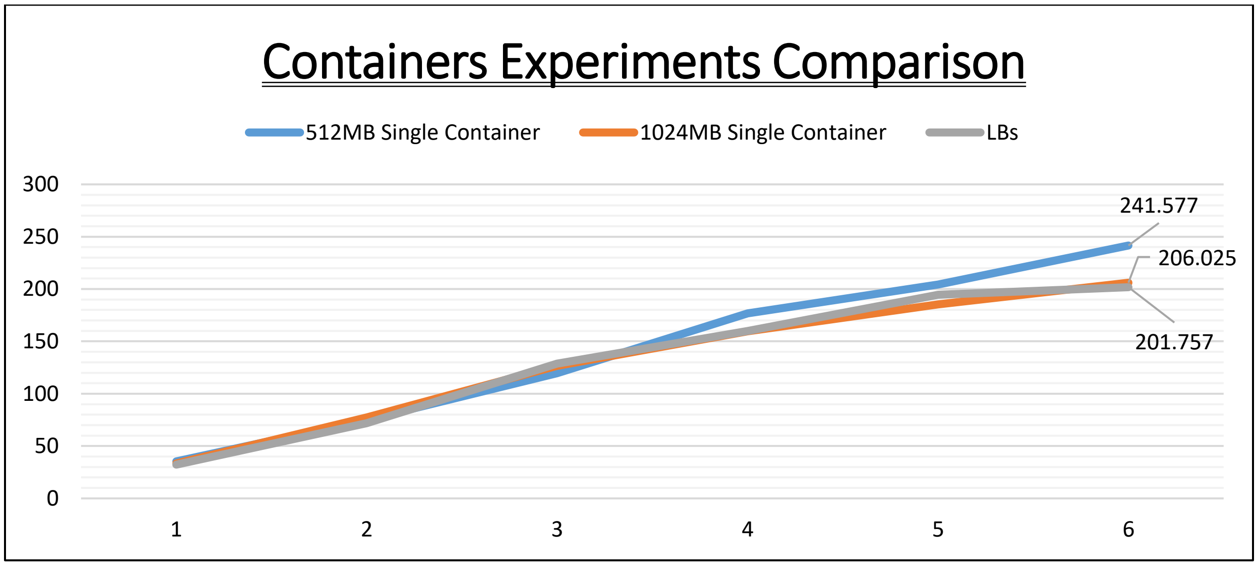

Figure 19: Containers Experiments Result

CHAPTER 5: CONCLUSION

5.1 Conclusion

This research presents an experimental analysis to investigate best possible method to introduce container technology within the network without incurring additional complexity and better response time for website with huge traffic and keeping cost low. This research presents round robin as a load balancer algorithm over HA-proxy webserver on container technology. Our experimental implementation of centralized physical machine reduces hardware cost, human intervention, power consumptions and space per site. Experiments clearly shows that these container technology is much efficient as compare to Virtual Machine as container response time is much lesser than virtual machines. Docker container scheme improves the utilization of hardware which reduces the cost and its maintenance.

5.2 Future Recommendations

The work contributed in this research can serve as a guideline for the future researchers who will be interested in and have an innovative approach towards the field of Cloud Computing with micro services like Docker container Technology.

Following research recommendations have been presented for further improvements in the proposed Self Organized Web Container algorithm.

- Test the environment over public network.
- Similarly, you can test the environment with different CMS containing database connections as well.
- Moreover, other types of web server can be incorporated to achieve better response time like Nginx.

References

[1] J. C. Dueˇnas, F. Cuadrado, B. Garc´ıa, H. A. Parada, and J. L.Ruiz, ″System virtualization tools for software development,″ Internet Computing, IEEE, vol. 13, no. 5, pp. 52–59, 2009.

[2] S. Seetharaman and K. Murthy, ″Test optimization using software virtualization,″ Software, IEEE, vol. 23, no. 5, pp. 66–69, 2006.

[3] G.-H. Kim, Y.-G. Kim, and K.-Y. Chung, ″Towards virtualized and automated software performance test architecture,″ Multimedia Tools and Applications, vol. 74, no. 20, pp. 8745–8759, 2015.

[4] L. M. Silva, J. Alonso, P. Silva, J. Torres, and A. Andrzejak, "Using virtualization to improve software rejuvenation," in Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on. IEEE, 2007, pp. 33–44.

[5] N. Bonfiglio, A. Ryan, Y. Zhang, and D. Mercer, "Systems and methods for on-demand deployment of software build and test environments," Apr. 18 2007, US Patent App. 11/788,219.

[6] S. I. Larimore, C. M. Murphey, and K. C. Obata, "Method and system for virtualization of software applications," Dec. 8 2015, US Patent 9,207,934.

[7] A. Aguiar and F. Hessel, "Embedded systems' virtualization: The next challenge?" in Rapid System prototyping (RSP), 2010 21st IEEE International symposium on. IEEE, 2010, pp. 1–7.

[8] M. Janssen and A. Joha, "Challenges for adopting cloud-based software as a service (SAAS) in the public sector." in ECIS, 2011.

[9] Csfi.com. (2016) Virtualization. [Online]. Available: http://www.csfi.com/index.php/virtualization/

[10] T. Burger. (2012, March 5) The advantages of using virtualization technology in the enterprise. [Online]. Available: https://software.intel.com/en-us/articles/the-advantages-of-using-virtualization-technology-in-the-enterprise

[11] B. Kirsch. (2014) Virtual machine. [Online]. Available: http://searchservervirtualization.techtarget.com/definition/virtual-machine

[12] N. Naik, ″Building a virtual system of systems using Docker Swarm in multiple clouds,″ in IEEE International Symposium on Systems Engineering (ISSE). IEEE, 2016.

[13] D. Merkel, ″Docker: lightweight linux containers for consistent development and deployment, ″ Linux Journal, vol. 2014, no. 239, p. 2, 2014.

[14] G. M. Tihfon, J. Kim, and K. J. Kim, ″A new virtualized environment for application deployment based on docker and aws,″ in Information Science and Applications (ICISA) 2016. Springer, 2016, pp. 1339–1349.

[15] C. Anderson, ″Docker [Software Engineering],″ IEEE Software, no. 3, pp. 102–c3, 2015.

[16] P. Marinescu, P. Hosek, and C. Cadar, ″Covrig: A framework for the analysis of code, test, and coverage evolution in real software,″ in Proceedings of the 2014 International Symposium on Software Testing and Analysis. ACM, 2014, pp. 93–104.

[17] W. Gerlach, W. Tang, K. Keegan, T. Harrison, A. Wilke, J. Bischof, M. D′Souza, S. Devoid, D. Murphy-Olson, N. Desai et al., ″Skyport: container-based execution environment management for multi-cloud scientific workflows,″ in Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds. IEEE Press, 2014, pp. 25–32.

[18] Docker.com. (2016) Docker use cases. [Online]. Available: https://www.docker.com/use-cases

[19] J. Turnbull, The Docker Book: Containerization is the new Virtualization. James Turnbull, 2014.

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

[21] Nitin Naik, ″Migrating from Virtualization to Dockerization in the Cloud: Simulation and Evaluation of Distributed Systems,″ in 2016 IEEE 10th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments.

[22] M. Resman, "CentOS High Availability," Packt Publishing, 2015, pp.2.

[23] S. van Vugt, "Pro Linux High Availability Clustering," Apress, 2014, pp.2.

[24] J. Lee, Tourrilhes, P. Sharma, and S. d S. d S. Banerjee, "No more middlebox: integrate Banerjee,integrate processing into network," SIGCOMM Comput. Commun.

[25] CITRIX [URL] 1999-2016 Citrix Systems http://docs.citrix.com/fr-fr/xencenter/6-5/xs-xc-intro-welcome/xs-xc-system-requirements.html

[26] Xiaoliang Li, Feng Liu, Zhenming Lei and Kun Zhang, self-organizing load balancing for network measurement server cluster," work is supported by National Natural Science Foundation of China. 2011 IEEE

[27] Zongjian He, Guanqing Liang, "Research and Evaluation of Network Virtualization in Cloud Computing Environment,"

[28] Virtualization [URL] May, 2016 https://en.wikipedia.org/wiki/Virtualization

[29] VMware Virtualization Technology [URL] Nov. 2007 [Online]. Available: http://www.vmware.com

[30] Unified Modeling Language [URL] Nov. 2007 [Online]. Available:http://www.omg.org/technology/documents/formal/uml.htm

[31] Jiaxin Wang , Lianhe Yang ; Miao Yu ; Wang "Application of Server Virtualization Technology Based on Citrix XenServer in the Information Center of the Public Security Bureau and Fire Service Department.' IEEE July 2011.

[32] Hardwar [URL] Jun, 2012 http://www.hardwaresecrets.com/everything-you-need-to-know-about-the-intel-virtualization-technology/

[33]LinOxide [URL] 2016, online https://linoxide.com/tools/configure-citrix-xenserver-6-5-vmware-workstation/

[34] Stephen Childs, Brian Coughlan, David O'Callaghan, Geoff Quigley, John Walsh, "A single-computer Grid gateway using virtual machines," 19th International Conference on Advanced Information Networking and Applications, 2005 IEEE.

[35] Bukhary Ikhwan Ismail, Ehsan Mostajeran Goortani, Mohd Bazli Ab Karim, Wong Ming Tat, Sharipah Setapa, Jing Yuan Luke, Ong Hong Hoe “Evaluation of Docker as Edge Computing Platform’ August 2015 IEEE Conference on Open Systems.

[36]Suchitra Venugopal Published on July 11, 2016 https://www.ibm.com/developerworks/cloud/library/cl-cloud-orchestration-technologies-trs/index.html

[37] Claus Pahl, “Containerisation and the PaaS Cloud” June 2015. IEEE Cloud Computing 2(3):24-31

[38] K. Boloor, R. Chirkova, T. Salo, and Y. Viniotis, “Heuristic-based request scheduling subject to a percentile response time sla in a distributed cloud,” in Proc. of IEEE GLOBECOM. 1–6, 2010.

[39] Advanced Bash-Scripting Guide [URL] 2014 onlineAvailable:http://www.tldp.org/LDP/abs/html/why-shell.html