Sandeep Aryal

Highly Available Web Service Using the Raspberry Pi Cluster

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

21 April 2017

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Sandeep Aryal Highly Available Web Service Using Raspberry Pi Cluster 34 pages 21 April 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Networking
Instructor(s)	Harri Ahola, Senior Lecturer

This project aimed to give theoretical knowledge of high availability and implementation of a highly available web service using open-source tools and the cluster of Raspberry Pi. In the project, high availability was demonstrated by implementing a web server and balancing the load in the Raspberry Pi cluster. The project was planned to be done in two parts.

Initially, the first part of the thesis was to do an initial setup which included initializing the Raspberry Pi, installing and configuring web server on two nodes and setting up one load balancer. Secondly, the project was driven toward the high availability achievement with the additional installation of a load balancer with a health check-up and failover. Additionally, a mirroring between the two web server nodes was done to sync the files and directories. Open-source tools like Raspbain, the Raspberry Pi operating system, HAProxy, Keepalived, Apache web server and Rsync were used to achieve the goal.

The design and implementation of the project demonstrated the high availability principle. The project was designed to create the prototype of cluster-based computing to gain high availability web service. As a result, highly available web service was successfully implemented using Raspberry Pi cluster and open-source software.

Keywords	Raspberry Pi, Cluster Computing, High Availability, Raspbian, HAProxy, Keepalived, floating IP

Contents

ABBREVIATIONS/ACRONYMS

3D	Three Dimensional
Bash	Bourne again shell
CESDIS	Center of Excellence in Space Data and Information Sciences
FAT	File Allocation Table
GUI	Graphical User Interface
HA	High Availability
HDMI	High-Definition Multimedia Interface
HPC	High Performance Computing
HTTP	Hypertext Transfer Protocol
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
Mac	Macintosh
MTBF	Mean Time between Failures
MTTR	Maximum Time to Repair
NASA	National Aeronautics and Space Administration
OSI	Open Systems Interconnection
RAM	Random Access Memory
RDP	Remote Desktop Protocol
RJ45	Registered Jack 45
ROM	Read-Only Memory
SAN	Storage Area Network
SD Card	Secure Digital Card
SQL	Structured Query Language
SSH	Secure Shell
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
VRRP	Virtual Router Redundancy Protocol
Wi-Fi	Wireless Fidelity

1 Introduction

In the present scenario, Information Technology (IT) companies and customers are concerned about performance, authenticity and availability. Investment in high-available complex clusters is in the peak in order to achieve the maximum number of nines. While manipulating the high load in the system, everyone is worried about the downtime and single point of failure. High availability is a characteristic of system design which can address later consideration.

The project objective is to introduce the high availability principle using the cluster computing mechanism. Furthermore, the implementation is based on open source tools and Raspberry Pi clusters to achieve highly available web service.

This thesis is written in five sections. The first section is the introduction to the topic. The second sections describes the theoretical background about the high availability principle, cluster computing mechanism and Raspberry Pi and Raspbian. The third section explains about the implementation of the project with the tools used for it. Similarly, the fourth section of the document emphasizes the result and the problem faced during the implementation. Finally, the last section presents the conclusion with benefits, drawbacks and recommendations.

2 Theoretical Background

2.1 Web Application Architecture

A web application is an application that can be gained by using a web browser. The client creates an HTTP request for a specific URL which maps to the resource on a web server. The web server processes the request and return the HTML page to the client, which the browser can display. The root of the web application is its server-side architecture. The application can consists of different tiers. The common example is three-layered architecture which consist of presentation, business and data layers. The goal of a web application architect is to minimize the complexity by separating the domain of security, high availability and high performance. [1]

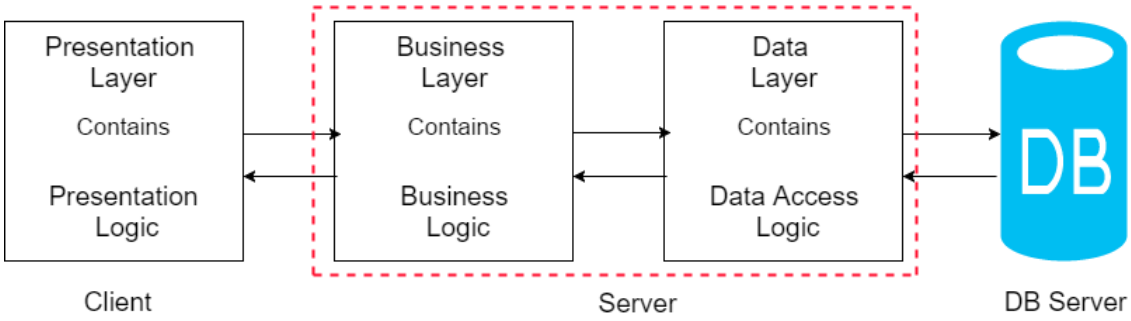

Figure 1. 3-Layered Architecture of Web Application. Reprinted from [2]

The presentation layer of web application architecture is concerned with UI and user synergy. The presentation layer consists of both server-side and client-side component. The server-side component renders the HTML and client-side component executes the scripts and displays to the browser. HTML, CSS and JavaScript are widely used technologies for the presentation layer. Similarly, the business layer of web application architecture, acknowledge the implementation of business logic and workflows. It sets the rules for processing the information and can associate multiple users. The business layer is also called middleware or backend. PHP, Python, Ruby and NodeJs are widely used technologies for the business layer in web application architecture. The data layer in web application architecture consists of physical storage for data persistence. The data layer consists of both data sets and the database management system or RDBMS software that manages and provides access to the data. [2]

As seen in figure 1, every tier can be operated or managed separately on different machines to achieve the high performance and high availability. High performance and high availability depends on the needs of an organization and what type of scalability they are doing, which is explained in section 2.4.

2.2 High Availability

In the present world, there might be a condition where the performance of the server may fall down. Either it can be caused due to a sudden rise in traffic or sudden power outage. The situation can be worse than we can imagine; it might be a full crash irrespective of the application hosted either in the physical machine or in the cloud. We cannot avoid such a situation. It is better to furnish the infrastructure in a way that it does not encounter a breakdown rather than hoping it will not occur.

These kinds of problem can be solved by using the high availability architecture and configuration. It is a way of designing and implementing modules and components of a system which assures excellent performance also in the time of heavy load. In spite of the fact that there is no specific principle for the implementation of high availability system, there are a few good practices which can be implemented to achieve the most out of the least resources.

2.2.1 System Downtime

System downtime is defined as the time period when a system, network or server is not available for use. It may be due to different reasons like hardware or power failure, software or operating system updates, system crashes, lack of network connectivity or hacker attack. Downtime can cause extensive loss in the company or an organization. All the services provided is kept on hold when they obtain downtime. For example, In August 2013, Amazon faced downtime for 15 minutes resulting a loss of 61187 euro per minute [3].

Downtime can be categorized in two types, scheduled downtime and unscheduled downtime. Scheduled downtime is the consequence of system maintenance which includes updating and upgrading the software, changing the database architecture or applying patches. However, unscheduled downtime occurs by an unexpected accident which includes a software and hardware failure. Unscheduled downtime can happen due to a problem in the power system or its failure. In general, scheduled downtime is not counted while calculating performance.

Table 1. Causes of downtime with examples. Reprinted from [4]

Causes of downtime	Examples
Planned downtime	Upgrades for firmware, drivers, operating system, software applications and hardware components.
Hardware failures	Broken storage unit, such as failed disk controllers and disk drives. Failed network component, such as switches, network cables and routes. Broken server infrastructure, such as power supply unit, system boards and memory chips.
Software failure or bugs	File exploitation, drive not responding, operating system stops of reboots time and again, malwares or viruses.
Maintenance or system outages	System needs to reboot or system board fails.
Human error	Intentional or accidental file or folder deletion, incompetence user, testing.
Local calamity	Storms, fires and other local disasters.
Regional calamity	Floods, hurricanes, earthquakes.

Table 1 lists the possible common causes of downtime and clarifies each with an example. We can see from the table that the factor of causing downtime can be both physical and natural.

2.2.2 Measurement of Availability

Availability, either low or high, is the calculation of the time when the system is working ordinarily. There is an equation for the availability calculation, which is given as:

$$A = \frac{MTBF}{MTBF+MTTR}$$

Here, A is the intensity of *availability* denoted as a percentage. Similarly, MTBF is the *mean time between failures*, and MTTR is the *maximum time to repair or resolve* a distinct problem. From the above equation, we can observe that MTTR and A are inversely related, A leads towards 100 percent when MTTR approaches zero. Similarly, increase in MTBF does not affect A more. As an assumption, if a system has an MTBF of 10,000

hours and MTTR of a single hour, it gets the availability of 99.99 percent. If we decrease the MTTR from an hour to 6 minutes, availability gets boomed by one additional 9, resulting in 99.999 percent. As a result, In order to get this figure of availability we would require an infrastructure which can run throughout the year with only 6 minutes of downtime. [5, p. 17]

2.2.3 The Myth of Nines

In theory, nines in reliability is the percentage of time a system is up and running. When the customer, management or the sales discuss availability or high availability, 'the nines' comes in the head of everyone. While considering availability, the nines of availability is always a trending topic as shown in table 2.

Table 2. Nines of Availability. Copied from [5, p. 11]

PERCENTAGE UPTIME	PERCENTAGE DOWNTIME	DOWNTIME PER YEAR	DOWNTIME PER WEEK
98%	2%	7.3 days	3 hours, 22 minutes
99%	1%	3.65 days	1 hour, 41 minutes
99.8%	0.2%	17 hours, 30 minutes	20 minutes, 10 seconds
99.9%	0.1%	8 hours, 45 minutes	10 minutes, 5 seconds
99.99%	0.01%	52.5 minutes	1 minute
99.999%	0.001%	5.25 minutes	6 seconds
99.9999% ("six 9s")	0.0001%	31.5 seconds	0.6 seconds

In table 2, nines indicates the uptime of the system in percentages over a given time period. Time period is usually calculates in a year. It is one of the common and easy ways to understand the availability and classify the system and its administrator ability.

The theory of nines is elementary and invalid where it does not count the system outage time. The nines of availability also assumes that all the time of system outage are of the same value to the organization, which is false. For example, according to table 2, 26 minutes of system outage in the Amazon web service give the 99.99 percent of availability if it is only outage in a year. It is a fact that 26 would affect the Amazon more, if it occurred on Sunday morning in late August when compared to a Friday night in

December. Similarly, if the system that handles the camera in the news room fails, it will affect more if it fails just before the broadcast rather after it. [6]

The ratio of complexity increases with the increase in the number of system components. We can take into account a system with ten components and an availability target of five nines i.e. 99.999 percent. Every component in the system has a 30 second responsible average downtime in a year. If either of the components fails for more than 30 seconds, another component must be liable for less. If one of the components gets down for five minutes, then the target will not be met.

2.3 Cluster Computing

A cluster is a generally utilized term which implies that independent computers are combined together into one system with the help of software and networking. In general understanding, when more than one computer utilized together to perform a task then it is recognized as a cluster. Clusters are mainly used for High Availability (HA) for greater reliability or High Performance Computing (HPC) in order to provide higher computational power than a single computer can cater for. The compositions of clusters are formed by many commodity computers which are interconnected by a high-speed dedicated network. Clusters can be categorized in various ways. [7].

Clustering was introduced in 1993 by two employees of National Aeronautics and Space Administration (NASA) named Donald Becker and Thomas Sterling. When Becker and Sterling were working in the Centre of Excellence in Space Data and Information Sciences (CESDIS) in 1994, the cluster project was named as Beowulf. [8] . After that, clustering has been practiced in companies and organizations with the goal to enhance the performance for scalability, high availability and reliability. As an example, Google combines more than 15000 computers to furnish a high-performance search engine. [9]

A cluster is made up of servers which are called nodes. The nodes are the part of cluster where they are connected via network. In order to get connected with network, nodes need to have the Network Interface Card for wired connectivity and a Wi-Fi adapter for wireless connectivity. The Network Interface Card enable nodes to communicate between other nodes and Storage Area Network (SAN) with in the cluster. There is no specific throughput requirement of the nodes although the capacity affects the general performance of the cluster. All the data that is required for the cluster is stored in an SAN.

Data in the SAN is available for every nodes in the cluster at same time or simultaneously. The storage unit demands better performance and decent components with the backup system in order to cope a single point of failure. [10]. There are different ways of architecting the clusters, which depend on the needs and requirements. In general, clustering can be categorized in the following ways.

2.3.1 High Availability (HA) cluster

High Availability (HA) cluster is a group of stand-alone computers where they work together in order to increase continuous availability of data or services. If a node fails, the service can be restored without affecting the availability of the services provided by another node in the cluster, and the process is known as failover. While the application will still be available, users may experience minimal interruption in service. In addition, there is performance drop due to the missing node. The function of these cluster is to guarantee that at least an instance of a service is running on a node at a time and if the node is not available, the service will failover to another node to achieve high availability. [11] .

High-availability clusters can be implemented for achieving the mission-critical application or services like databases, email, print, file, web or application servers [12] .

2.3.2 Load Balancing Cluster

Load balancing cluster distributes incoming requests for resources or content among multiple nodes running the same programs or having the same content. It is achieved by installing a service in multiple nodes which are configured to share the workload. Load balancing cluster gets ingress request and redirects them to the nodes as necessary. There is different algorithms to control the ingress traffic in the in cluster. A few widely-used algorithms are Round-robin, Weighted round-robin, Least-connection and Least-based. In addition, some load balancer is able to detect the failure. It keeps communicating with the nodes and stops forwarding the traffic after the failure. [11].

2.4 Vertical and Horizontal Scalability

Scalability is defined as the ability to handle the growing amount of work in an efficient manner. It is an ability for a system, network or process. It is general term which can

used in hardware systems as well as programs. In order for system to handle more work, either a power of the system can be increased or new system could be added. Scalability can be achieved in two ways, i.e. horizontal scaling and vertical scaling.

Vertical scaling or scaling up means the adding of more resources on top of existing system or node. For example, it can be adding more CPU, Disk Space or the memory. In order to get the more storage, more storage unit can be added. Irrespective of the scalability and power of the system, vertical scalability has a limit where the system cannot be expanded beyond the particular limit. This leads to the failover of the traditional approach of handling the big data. One application of vertical scaling is by making the system powerful by adding the resources and creating the multiple virtual machines. [13]

In other hand, horizontal scaling means adding more system or nodes rather than making powerful nodes. The term scaling out is also used for horizontal scaling. There is a linear correlation between the number of system and the performance. For example, if 'X' amount of data can be processed in certain time, in order to process '2X' amount of data addition node is required. This shows that the linear scaling comes handy and affordable, as commodity hardware can be used. Horizontal scaling can be achieved by clustering, distributed file system and load-balancing. It works in distributed systems model which needs special software or programs designed to take the advantage of distributed system. Program like Hadoop, mongo dB etc. are designed to work in distributed system. In this thesis, horizontal scalable is done in order to achieve the high availability. [13]

2.5 Raspberry Pi

Raspberry Pi is a small computing device which makes it perfect for portability, where a small unit can be placed anywhere in a house and can be bought at as low as 30 to 45 euro. It is powerful and hackable, so it can be used for testing and research. Raspberry Pi is available in two different models i.e. Model A and B. Model B has many connectors compared to Model A, whereas Model B is more expensive than Model A. [14]. Raspberry Pi is being widely used in a variety of projects like home security, robotics and sensors within schools, colleges and universities.

Broadcom is a manufacturer of a computer's chipset collaborated with University of Cambridge in the United Kingdom to co-find the Raspberry Pi Foundation. The foundation is a charitable organization dedicated for educational research. Initially, Raspberry Pi was designed for kids to learn programming and for the citizens of the third-world countries

to have cheap and fully functional computers. Later the concept was evolved when computer enthusiastic found different innovations using the Pi. Over the years, there have been several versions released like B, B+, 2 and the latest version 3 B. The capacity and performance of the Pi is being increased with each update. RAM has increased from 256 MB to 1 GB and CPU has been bumped from 700 MHz single core to 1.2 GHz quad-core. The number of USB ports, GPOs and other specifications have also increased. To summarize, all these updates provide decent computing power to the latest version of Pi. [14]. Raspberry Pi 2 Model B, which is used in this project, has a 900 MHz quad-core ARM Cortex-A7 CPU and one GB RAM.

2.6 Raspbian and Debian

Raspberry Pi is only a piece of PCB without an operating system. There are different operating system which fits the Raspberry Pi, counting Pidora, RISC OS, Arch Linux, and Raspbian. At present, Raspbian is the most popular operating system for Raspberry Pi. It is Linux based and open source in nature. Raspbian is made by the modification of Linux Debian distribution. Raspbian adds a variety of software packages to Raspberry Pi which makes it easier to use and can be operated out of the shelf. Raspbian is the endorsed operating system for novice to start with. [15].

The Debain is an authentic distribution of Linux which was built in August 1993 by Ian Murdock. Being the part of Debian, Raspbian inherit almost all the tools and features, including its enormous repositories of software bundles. There are different components which makes the advanced Linux distribution. All these different components add the modern features computing with Pi. Some of the main components are mentioned in the following sub-sections.

2.6.1 The Raspberry Pi Bootloader

Bootloader is the sets of codes which run when the hardware is turned on. It initializes the hardware to a known state. It confirms that there is nothing missing in the files and everything is working perfect. After the initialization, the process of loading the Linux kernel begins. In Raspberry Pi, there are first and second stage bootloaders who does

the above mention task. The initial stage bootloader is coded into the ROM by the manufacturer and cannot be altered. Similarly, second stage bootloader is coded in the SD card which is run by first stage bootloader.

2.6.2 The Linux Kernel

The Linux Kernel is the foundation of Raspbian where it serves as the core of Linux distributions for servers and desktop environment. It is also used in android and embedded system. In Raspberry Pi, it guides every operation from receiving the keystroke when something is typed on keyboard to displaying output to monitor using HDMI. The Linux Kernel was made by Linus Torvalds. Linux kernel provides the layer for hardware and software application in order to perform the task. For example, if someone is trying to connect Wi-Fi adapter, at first Linux kernel needs to know about the about the Wi-Fi adapter and the way to use it. Only after the kernel recognises the Wi-Fi adapter the user can connect to the Wi-Fi. Another good example is, when the USB drive is plugged in, the kernel automatically finds it and informs the daemons, which makes the files available instantly. Currently, the wide variety of devices available on the markets and recognised by the Linux kernel, with more being added endlessly. After the successful initialization of the kernel, it runs the program called *init* which completes the initialization of the Raspberry Pi. Consequently, the daemons will start loading in the background followed by the graphical user interface.

2.6.3 Daemons

Daemons is the piece of software or program which runs secretly in the background. Linux runs lots of daemons at a time which welcomes the request for services coming from other programs and hardware and also from the other computers in the network. Some examples of daemon are Cron, a job scheduler, Apache web server, Autofs a daemon which mount removable storage device. [16].

The Linux distro Raspbian is not functional only with kernel. It also demands additional software which allows the user to reach kernel and to handle the operating system. The collection of programs and scripts bundled in core operating system makes it possible.

2.6.4 The Shell

The shell is lunched by *init* after loading all the daemons. A shell act as an interpreter, where it allows Raspberry Pi to monitor and control using the commands. Despite of the fact that it looks like decades ago computer, it is one of the most powerful parts of Raspbian. There are different shells in different Linux distro. Bourne Shell, Korn Shell, C Shell, Bash Shell and tcsh are few examples. In most of the Linux distribution, Bash is the default and widely used shell which comes also in Raspbian as a default shell.

Bash is a sh-compatible shell which combines the most effective features from the C shell and Korn shell. It adds features like command line editing, job control, shell functions and laisse, unlimited size command history, unlimited sized indexed array and integer arithmetic in any base from two to sixty-four which makes Bash interactive and programming friendly. In addition, it has ability to run scripts without any modification. [17].

2.6.5 The Desktop Environment

It might be difficult for a beginner to use a computer without the graphical user interface. A GUI or desktop environment provides user the environment beyond keyboard and black and white screen which includes viewing videos and pictures, browsing internet and similar other things. Raspbian comes with a graphical user interface called PIXEL which stands for Pi Improved Xwindows Environment, Lightweight. It is designed in way that it can run with minimum resource.

3 Design and Implementation

The project was designed and completed in two parts. In the first part, the basic initial configuration was done, which included installing the Raspbian, configuring the hostname and connecting to the networks. Similarly, two nodes of the web server and single node load balancer were also configured in first phase. Mirroring the files and folders between the web server nodes and adding one more load balancer with a failover and health check-up was done in the second phase.

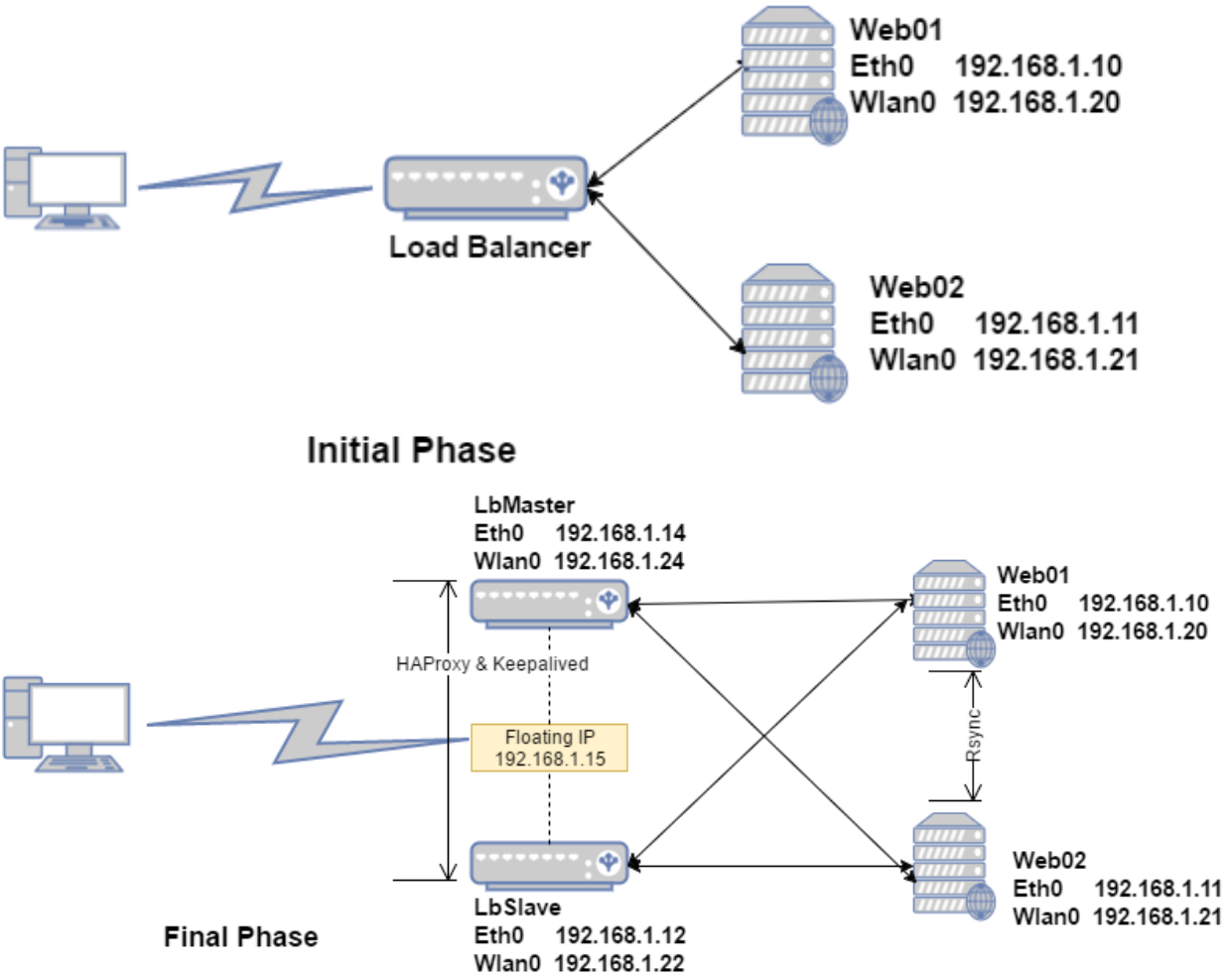

Figure 2. Initial Vs Final Phase Topology Diagram

Figure 2 shows how the project was planned and designed in two phases. In the second phase, one more load balancer was added with advanced features to withstand a failover and with the ability of a health check-up. In addition, web server nodes were synchronized to share updated files and folders. The topology shown above and features were added following the principle of cluster computing to obtain high availability.

This section of the document explains in detail how the project was designed and implemented. The project was designed using the following devices and software.

- Raspberry Pi 2 model B as the main computing equipment
- LAN cabal and wireless dongle to connect to a network
- Switch/Router for connecting different nodes of the web server and load balancer
- Apache for running the web service in different nodes
- HAProxy for load balancing

- Keepalived for health check-up and monitoring the automatic failover
- Rsync, SSH and Corn Job for mirroring the files between the web nodes.

3.1 Initializing the Raspberry Pi

Raspberry Pi is the main component of this project where Raspberry Pi 2 model-B was used because it was available in the lab. Besides easy availability, it also meets the basic performance power needed for the project with four processors and 1 GB of RAM. Raspberry Pi comes without an operating system and the storage media. Raspberry Pi has the slots for Micro SD for loading the operating system. The process was started by creating the bootable Micro SD with Raspbian Jessie Lite, whose image is available on Raspberry Pi official website. Raspbian Jessie Lite is the image based on Debian Jessie. The Lite version was chosen in comparison to PIXEL because the PIXEL version provides GUI which was not needed for the project. The decision was taken to avoid the heavy unnecessary component.

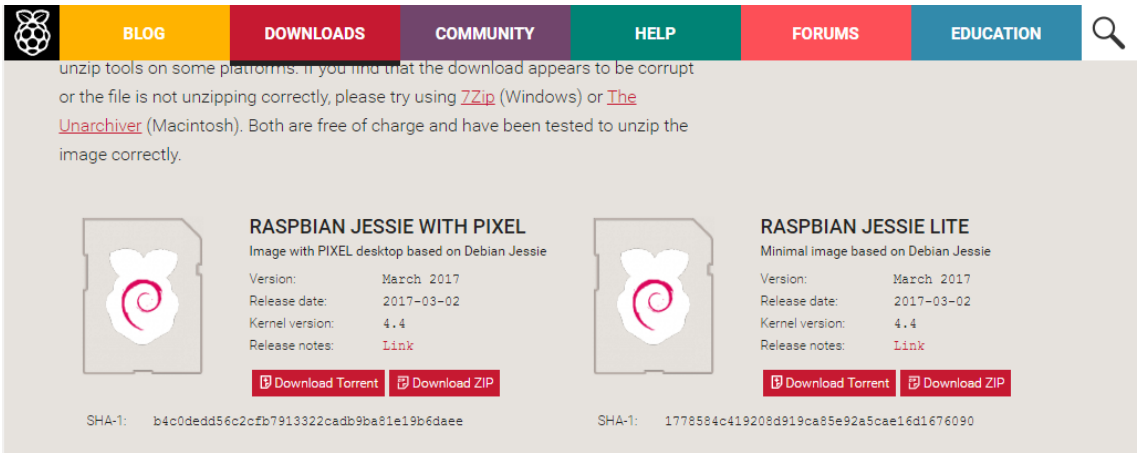

Figure 3. Raspbian Download Page

Figure 3 is a screenshot from the Raspberry Pi official download page. We can see in the figure that there are two options available. One on the right is with the GUI interface and the other one on the left is lite version without GUI.

3.1.1 Unzipping the Image

The Raspbian image available on the Raspberry Pi website is archived in ZIP format. In order to unzip the image file, unzipping tool was used. There are different unzipping tools available for different operating system. 7-Zip, The Unarchiver and Unzip are the popular unzipping tools used in Windows, Mac and Linux platform respectively. As Mac was used to create the bootable Raspbian image, the Unarchiver was used in this project.

3.1.2 Writing an Image to the SD Card

A secure digital (SD) card is the type of storage medium with high performance and portability. It is available in different ranges from phones, cameras to PCs. The Raspberry Pi has the SD card slot and it allows to install an SD card, which works as the storage tool similar to SSD, Hard Disk or USB storage. Due to its smaller size and high performance capacity, the SD card makes it best suit for embedded devices. [18, p. 19].

Installation of an image was done in two steps. Initially, it started with formatting the SD card to the FAT file system and then copying and pasting the image. FAT is the mechanism which tracks if the sectors in the disk are full or free to be written. FAT was developed in the 1970s and was used in floppy disks. Due to its strong architecture and ingenuity, its continuity is found in the SD cards till today and It was used to install the Raspbian. [18, p. 21].

Most of the SD cards available in the markets are already formatted to FAT, as FAT is popular among handheld devices. However, formatting once own is recommended as it might not be in ready to use state. SD Association has developed software for formatting an SD Card named SD Card Formatter which is available for MAC and Windows.

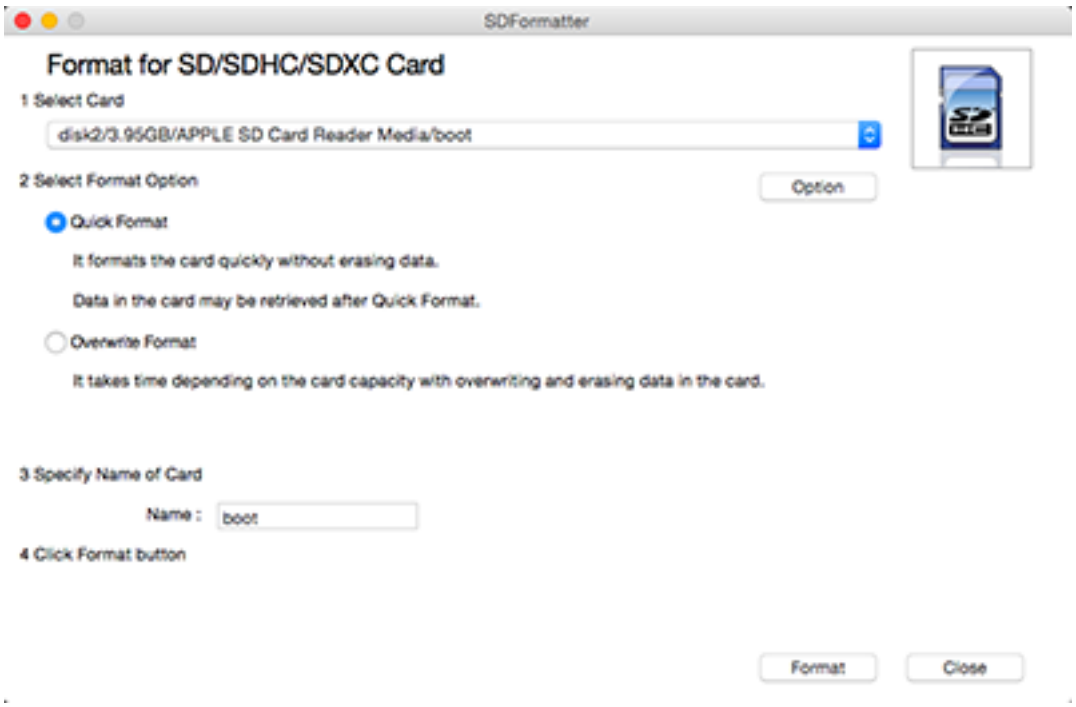

Figure 4. SD Formatter for MAC

Figure 4 is a screenshot of the SD Formatter software for Mac. It has the option to select the card from the available list. Lists are available according to the number of SD cards connected to the Mac. Secondly, it has two options, a quick format and an overwrite format. The quick format formats the card quickly without erasing the data whereas overwrite format formats the SD card erasing the data and it takes time depending upon the read-write speed of the SD card.

After successfully formatting the SD card, the image files were copied onto the SD card. The process of copying the image onto the SD card is different according to the host operating system. There are also many tools available to the process. Freely available applications can be used to avoid the difficult commands. These tools are helpful for a novice who does not have the experience of command line interface. One of the popular applications is Etcher, which is available for all three platforms i.e. Windows, Linux and Mac.

Figure 5. Etcher, copying image to SD card

Figure 5 is a screenshot of Etcher. It provides an easy graphical interface to the user to copy the image onto the SD card. It is an open-source tool. The process of copying the image starts initially by selecting the image from the disk drive, secondly selecting the drive i.e. SD card and finally, writing it to the disk by clicking the flask button.

3.1.3 Enabling the SSH

The Secure Shell, SSH is the software defined access to unsecure network. It is a powerful and popular protocol to access the remote machine. The connection between the SSH client and server is secure and encrypted, which makes it efficient enough to be used in mission-critical applications. SSH works on client-server design, where the server welcomes or denies the incoming connections to its host and the host sends the request to its server. In spite of the fact that SSH stands for Secure Shell, it is not a shell or a command interpreter, it does not provide command history and wildcard expansion. Rather it provides the infrastructure for running a shell on a remote host. Authentication, integrity and encryption are provided by SSH protocol to the transmitted data over the network. [19, pp. 1-3].

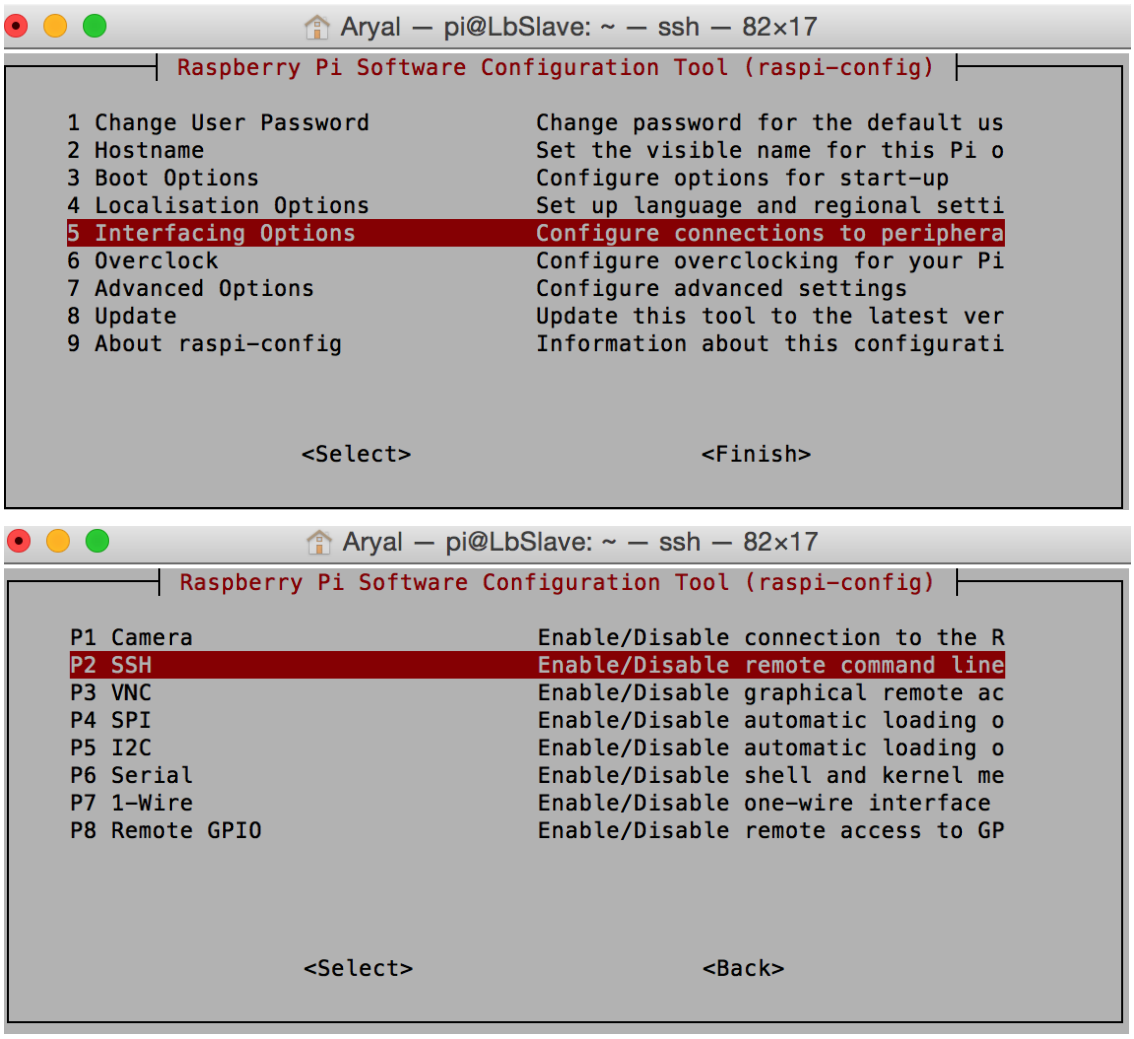

Figure 6. Enabling SSH, raspi-config

Raspbian comes with the SSH installed but it is disabled by default for security. In the project, SSH was enabled in all the Raspberry Pis to establish a secure and ease connection. In Raspbian, *raspi-config* provides the interface from where SSH can be enabled or disabled according to need. Figure 6 is a screenshot of the interface which comes after entering the *raspi-config* command in the terminal.

3.1.4 Changing the Hostname

As there were four Raspberry Pis used in the project, it was difficult and confusing to determine the role of the Pi because all of the Pi's default hostnames were *pi@raspberry.* In order to make the role of the Pi easy and clear, the hostname of each Pi was changed according to the role.

Table 3. Roles of Raspberry Pi and Hostname

Raspberry Pi No.	Role	Hostname
1	Web server node 1	Web01
2	Web server node 2	Web02
3	Primary load balancer	LbMaster
4	Secondary load balancer	LbSlave

Table 3 clarifies the role and hostname of the Pi. The first two Raspberry Pi acted as the web server nodes where there hostname was assigned as Web01 and Web02 respectively. Similarly, Pi three and four acted as the primary and secondary load balancer where their hostname was defined as LbMaster and LbSlave respectively.

In Debian Linux, the hostname can be changed temporarily and permanently. As the requirement was to change it permanently, the permanent hostname was achieved by editing the two files which were */etc/hosts* and */etc/hostname.* These two files were accessed by using the Nano editor and the old default name 'raspberry' was replaced with new hostname as mention in the table above. The process was completed by saving the files by running the command */etc/init.d/hostname.sh* and rebooting the pi.

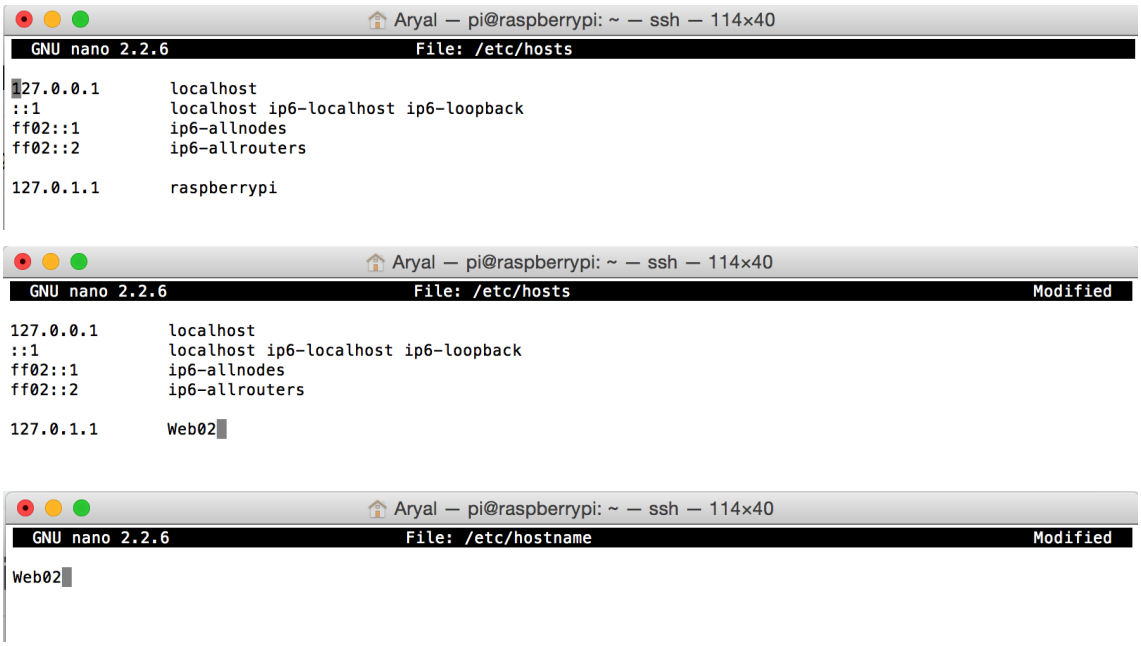

Figure 7. Changing the Hostname

The sequence of the picture above in figure 7 shows how the hostname was changed permanently. At first, the default hostname 'raspberry' was changed to 'Web02' in the file */etc/hosts* and then default hostname 'Web02' was added to the file */etc/hostname*. Similarly, this step was followed for all of the Raspberry Pi nodes and given the hostname as listed in table 3.

3.2 Networking

Networking or network configuration was one of the important parts of the project. Almost everything in the project was tied up with the network. All the four Raspberry Pis, two nodes with the web server and another two nodes with the load balancer are communicating with each other through the network. So, network planning and configuration played a vital role in the successful completion of the project. This section of the document will clarify the network-related plan and configuration in detail.

Raspberry Pi 2 model B, which was used in this project, comes with network interface where we can plugged in RJ45 and make it available for the network. An external wireless adapter was connected to its USB port in order to gain wireless connectivity. Wireless connection is not reliable compared to a wired connection, so a wired connection was taken into account during configuration of the nodes. A wireless connection was used only for the configuration which helped the admin to get rid of LAN cables and made the project portable to carry between places. A wireless interface was also used for testing purposes.

Table 4. IP Address Planning

Hostname	Interface	IP Address	Subnet	Default Gateway
Web01	Eth0	192.168.1.10	255.255.255.0	192.168.1.1
	Wlan0	192.168.1.20		
Web02	Eth0	192.168.1.11		
	Wlan0	192.168.1.21		
LbMaster	Eth0	192.168.1.14		
	Wlan0	192.168.1.24		
LbSlave	Eth0	192.168.1.12		
	Wlan0	192.168.1.22		
Floating IP		192.168.1.15		

Table 4 shows the IP address planning for the project. The IP address was chosen from the 192.168.1.1/24 network as it made the calculation easy. We can see that every node is in the same network so that they can communicate easily. In the table 4, eth0 means the Ethernet interface, it can also be recognised as wired connection and Wlan0 refers to the wireless interface or wireless connection. All the nodes have the default gateway 192.168.1.1 which is the IP address of the router.

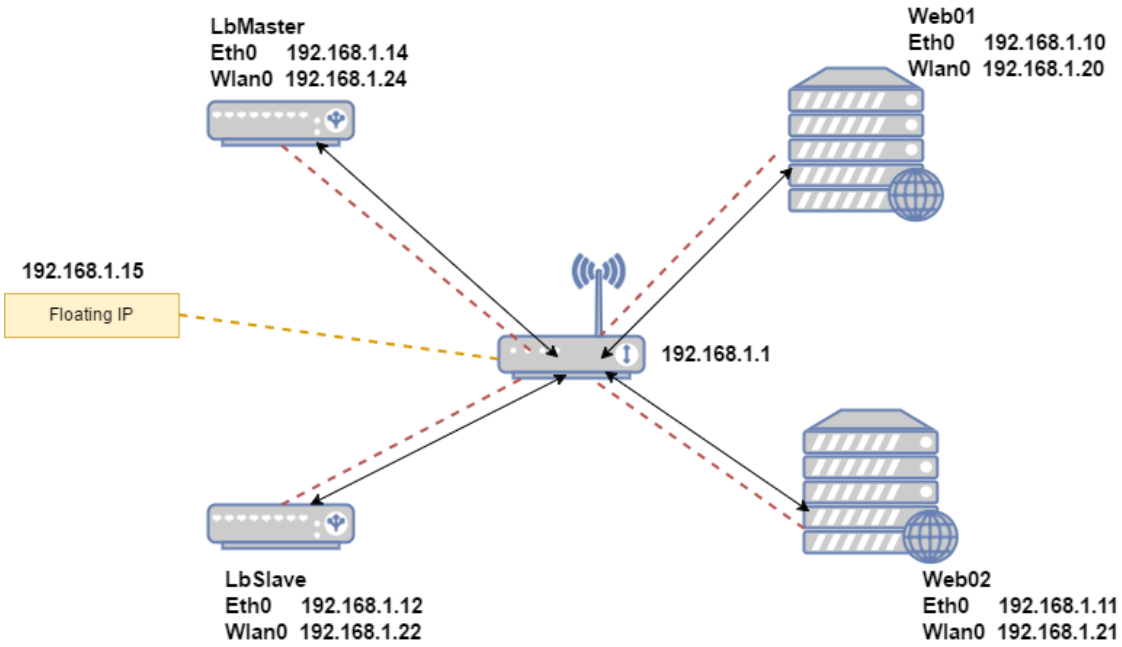

Figure 8. Topology with IP Address

Figure 8 is the network topology diagram. Black solid lines refer to the wired connection which is associated with the Eth0 interface and red dotted lines show the wireless connection associated with Wlan0 interface. Similarly, the yellow dotted line is the virtual link to the floating IP.

3.2.1 Configuring the Static IP

A static IP address is the IP address which remains the same until and unless an administrator wants to modify it whereas dynamic IP address changes time and again and is

leased by DHCP server. Static IP addresses are also referred to as dedicated IP addresses or fixed IP address. The router that was used in the project supports the leasing of IP addresses as it has a built-in DHCP server but getting different IP address time and again affects the configuration done on web server nodes and load balancer. In addition, it also affects the SSH which adds an extra connection of Raspberry Pi nodes to the monitor to know the new IP address. A static IP address was configured by adding additional lines in the file */etc/dhcpcd.conf*.

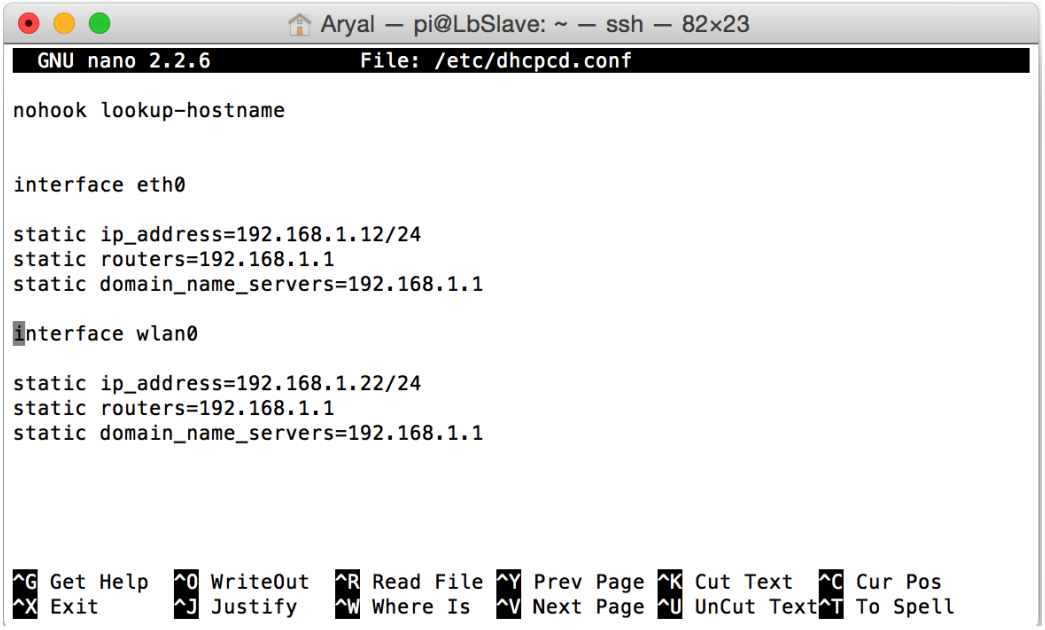

Figure 9. Static IP Address Configuration

Figure 9 shows how the static IP address was configured in LbSlave i.e. load balancer slave. It can be seen that a static IP address for eth0 interface was set to 192.168.1.12 and wlan0 was set to 192.168.1.1 where both has the subnet of twenty four. Eth0 interface refers to the wired connection and wlan0 interface refers to the wireless connection. Both the interfaces has the static route of 192.168.1.1 which is the IP address of the central router.

3.2.2 Connecting to a Wireless Network

As mentioned above, wireless connection helped to make the project portable and it was used for testing and configuration purpose. All the SSH connections were also estab-

lished through wireless network. Wireless connection gave freedom in a way. A connection to the Raspberry Pi nodes can be made from any device which is connected to the same wireless access point. As most of the devices come with the Wi-Fi support, even a mobile phone was used to check the status of the nodes.

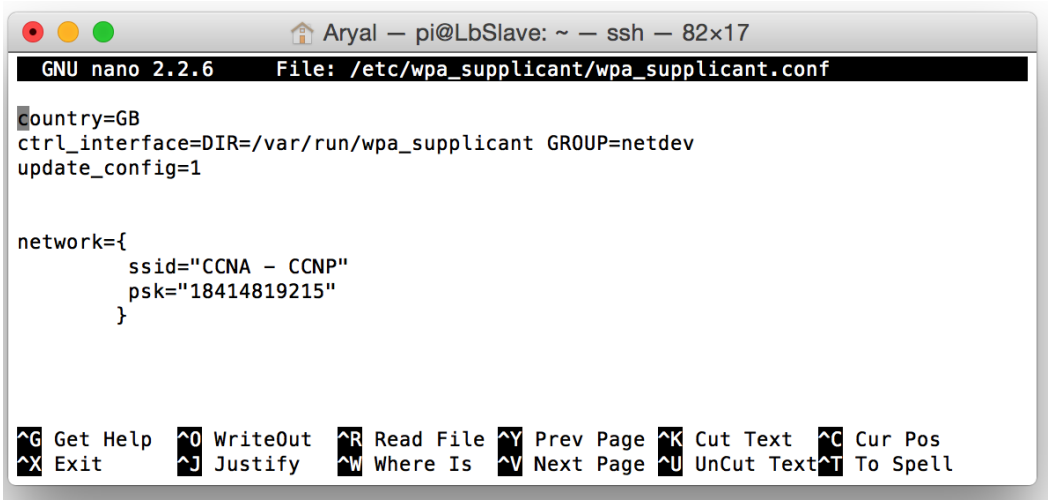

Figure 10. Connecting to Wi-Fi

Configuration of wireless network was done by adding the wireless network credentials to the file */etc/wpa_supplicant/wpa_supplicant.conf.* Figure 10 shows how the Wi-Fi credentials were added. In the figure, SSID refers to the name of wireless network where its name was 'CCNA - CCNP' and psk means the password where '18414819215' was the password for the wireless network. Finally, the wireless connection was established.

3.3 Understanding and Configuring HAProxy

HAProxy is the short form of High Availability Proxy. It is an open-source and trendy load balancer which runs on layer seven of the OSI model i.e. application layer. It can be used to achieve the reserve proxy for TPC and HTTP-based internet services. [20]. It distributes the load across the different servers running web, application and database which makes the server infrastructure more reliable and better performer. Due to its features of using low memory, single-process and event-driven model it can handle the large amount of simultaneous traffics. It is currently available for Solaris, Linux and FreeBSD and being used in high-traffic websites like Alibaba, GitHub, Imgur, Instagram, Stack Overflow, Tumblr, Twitter and Vimeo. [21].

3.3.1 Load Balancing Algorithms of HAProxy

There are different ways and algorithms which can be used to distribute the load from the client wisely. The algorithms can be configured in a group or stand-alone depending upon the type of service and available pool of servers. A single load balancing algorithm can be used at the time of normal traffic and a group of algorithms can be used to distribute the request under service stress. HAProxy provides a variety of load balancing algorithms. Some of the mostly used algorithms are listed below.

- Round Robin
  It is the simple technique of load balancing where load is distributed to the server in-turns according to its weight. After it reaches the end node, the process will start from the first node. Round robin is suitable for servers with identical capacity and configurations. The weighted round robin or static round robin accounts for the weight allocation to forward the request. It mean changing the weight in real time has no effect whereas a dynamic round robin accounts for the real-time weight.

- Least connection
  In this algorithm, the server having the lowest number of connections will get the new incoming requests. The least connection algorithm is mostly used in the scenario where long sessions are expected like TSE, SQL, and LDAP. However, it does not fit better for protocols like HTTP which uses short sessions. Its dynamic nature adds the feature to adjust the weight in real time.

- Source
  This algorithm hashes the source IP address and designates it to one of the server from the pool which is available at that time. This assures that the source client's IP will always be bound to the same server until and unless the server is up or down. It provides the best-effort stickiness to the source and it does not require any cookie. The Source algorithm is static in nature but it can be modified.

- uri
  In this algorithm either the whole URI or the left part of the URI before the question mark is hashed. It distributes the hashed value according to the total weight of the running web servers, which assures the same URI to bind to the same server

as long as the server stays up and running. It helps to expand the cache hit when used with a proxy cache and anti-virus proxies. It is static in nature but can be modified as a source algorithm. In addition, uri supports “len” and “depth” parameter where its value is a positive integer. These parameters may help to balance servers based on the former part of the URI only. The “len” indicates the numbers of characters from the beginning of the URI to hash. Similarly, the “depth” indicates the maximum depth of the URI to hash.

- url_param
  This argument looks after the URL parameter that is specified in HTTP GET request. It tracks the identifiers of the user and makes sure that same ID will be sent to the same saver. If there is a change in the status of the server, a round robin algorithm will be used.

- hdr
  Hdr is the short term for HTTP header. It looks up for the matching HTTP header in every HTTP request. As in the function ‘hdr()’, the parentheses consist of the header name which is not case-sensitive. The round robin algorithm comes in effect if the header is empty. In an option ‘use_domain_only’ reduces the hash algorithm to main domain with specific header such as ‘host’. For example, while taking the host value “tuubi.metropolia.fi”, “tuubi” will only be acknowledged. It is static in nature but can be modified as the source algorithm and uri.

- Rdp-Cookie
  The RDP cookie inspects and hashes every incoming TCP request. It is appropriate as a diminished persistence mode which is able to send the same user to the same server or the same session. If the cookie is found missing, the round robin algorithm will be used. It requires the configuration of the frontend assuring that RDP cookie is available in the request buffer. It is static by default and can be modified using “hash-type”. [22].

3.3.2 Configuration of HAProxy

In the project, HAProxy was chosen to work as a load balancer. In the topology, load balancer is configured between the client and the web server nodes. It keeps track of the web service and the health status. If the web server crashes it distributes the traffic to the web server node which is online. HAProxy is available in the package, which was

installed in both the load balancer nodes i.e. LbMaster and LbSlave by entering the command *apt-get install haproxy -y*. The configuration file of the HAProxy is situated in the directory */etc/haproxy* and the name of the configuration file is *haproxy.cfg*.

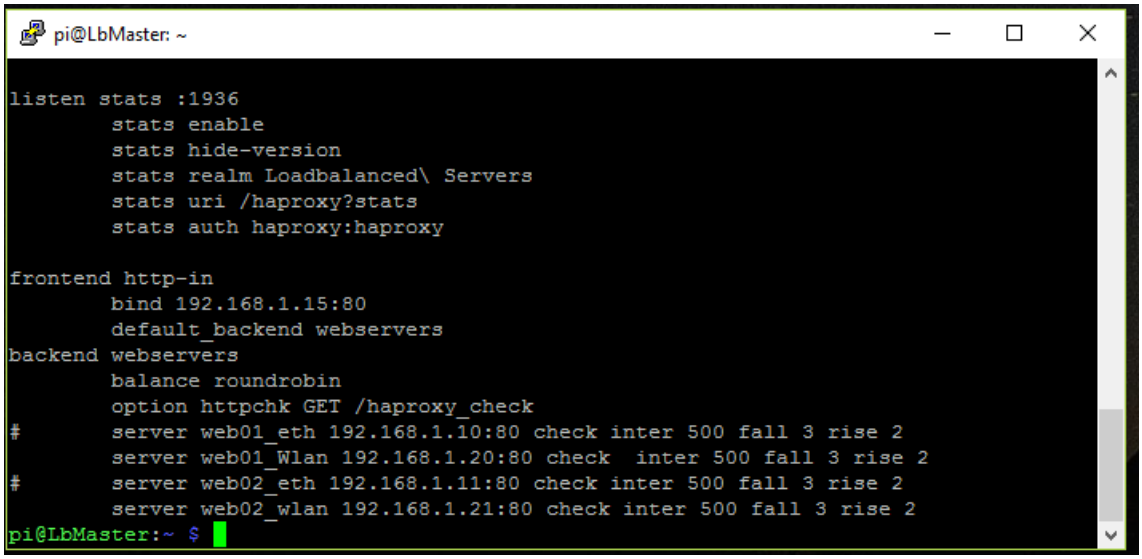

Figure 11. HAProxy Configuration for LbMaster

Both nodes of the web server have a similar configuration for HAProxy. Only the screenshot of LbMaster was shown and explained to avoid repetition. Figure 11 shows the HAProxy configuration for LbMaster where it is running as a daemon. There is a default block in the HAProxy whose configuration will be applied to all other blocks. There is HTTP mode TCP mode in HAProxy, where HTTP was selected which helped to make the reliable connection to web server. The frontend block in the HAProxy is the entry point for all the incoming traffic. The frontend was configured by giving it the name 'http-in' and binding it to the IP address 192.168.1.15. Here, the above mentioned IP address is the floating IP which floats between two load balancer nodes and 80 is the port for the http connection. Every incoming traffic coming to the floating IP was redirected to the cluster of backend webservers with the statement 'default_backed webservers'. All the nodes in the web server cluster are listed in the backend blocks with their IP address, load balancing algorithm and health check-up status. In the project, 'roundrobin' algorithm was used and both interface i.e. wired and wireless of both the web server nodes are add to the backend. HAProxy was configured in way that it checks the health of each nodes of web server in every five hundred milliseconds. If the health check-up fails for three times it will make the node down and the node will go to up state from down state if health check-up succeeds for two intervals.

Similarly, statistics on HAProxy was configured by adding the listen block. It was configured by giving the name ‘stats’ followed by listen tag where statistic will be listened to port 1936. The version was made hidden in order to prevent the version related exploits. The URI was added to set the exact location of the statistic page and finally the username and password were set.

3.4 Keepalived for Failover and Health Check-up

In the project, HAProxy was being used as a load balancer where the goal was to achieve high available service. One load balancer was not enough to achieve the high available service. To make the service more reliable multiple load balancer can be used as a backup in case of failover of the one. In this project, two nodes of HAProxy was used to achieve the high availability. However installing two nodes only was not enough. There needs to be communication between the load balancer nodes to know each others status. Keepalived is the open source software which was used in the project to detect the failover between two load balancer nodes to gain high availability [23]. Keepalived uses keepalived daemon to check the health and VRRP for detecting failover. Here, keepalived was used for the both failover detection and health check-up.

VRRP elects one node as a master from the pool of participating nodes. Other secondary nodes actively listen to multicast packets sent from the high priority node. The backup node becomes the master and appoint the configured virtual IP to it, if the secondary node does not receive the VRRP advertisement for three instances of the configured advertisement timer. If there are more than multiple backup nodes, the one with the highest priority will get the IP.

3.4.1 Topology Design and Working Principle

This section of the documents explains about the topology design of Keepalived and how it was implemented between the HAProxy nodes. In initial phase, single node of load balancer was configured which does not required keepalived in action. In second phase, keepalived played the role of failover detection and health check-up between the load balancer nodes.

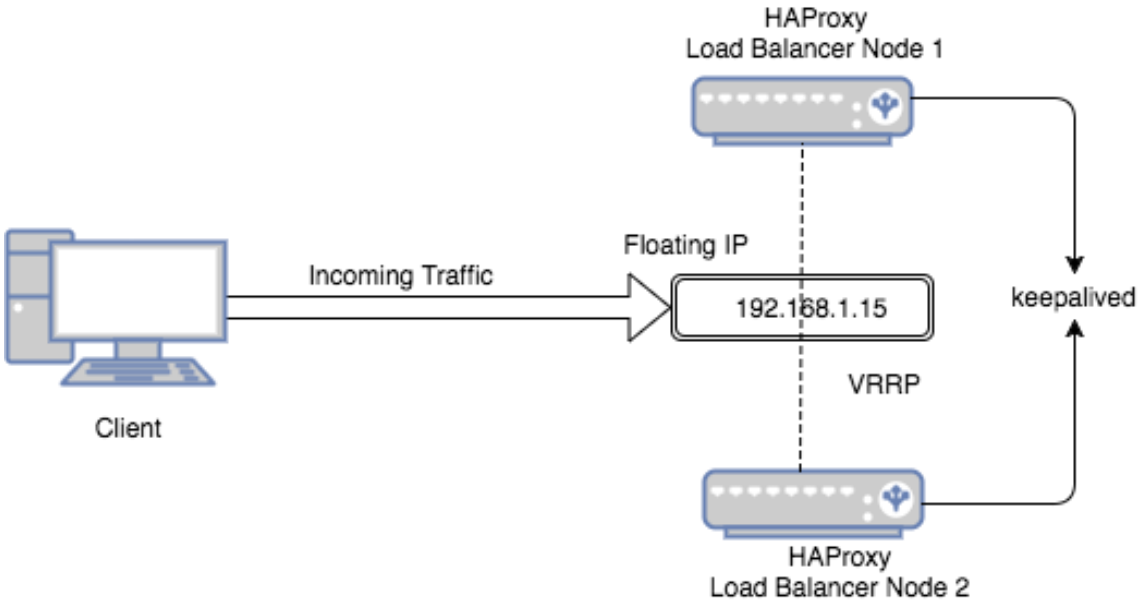

Figure 12. Load Balancer with Keepalived

As shown in the figure 12, keepalived was configured in both nodes of HAProxy load balancer. Floating IP or virtual IP also comes in action while configuring keepalived. The client is only aware of floating IP, which is used to access the web server behind the load balancer. Keepalived uses the VRRP to check the failover where floating IP floats to the active nodes resulting high availability. At initial condition, traffic gets inside through node one as it is configured as the master node with high priority id which sticks the floating IP to it.

3.4.2 Installing and Configuring Keepalived

Firstly, keepalived software was installed in both load balancer nodes. It is available in a package which was installed by writing the command *sudo apt-get install keepalived –y*. Keepalived configuration is done by adding necessary script in the file */etc/keepalived/keepalived.conf*. A Nano editor was used to add the script.

```
vrrp_script chk_haproxy {
          script "pidof haproxy"
          interval 2
}

vrrp_instance VI_1 {
```

```
        interface eth0
        state MASTER
        virtual_router_id 77
        priority 101
        virtual_ipaddress {
                192.168.1.15
        }
        track_script {
                chk_haproxy
        }
}
```

Listing 1. Keepalived Configuration Script of LbMaster

Listing 1 shows the configuration script of LbMaster. In the list above we can see the script and the instance. The script will check if HAProxy is running or not. It will keep the node as a master if HAProxy is running and if not, VRRP will communicate with another node make another node as a Master and current node as backup. With the movement of master and backup, floating IP also moves accordingly. In the script above, the name of the script was defined as 'chk_haproxy' which will check the status of HAProxy in every two seconds with the help of process id. Instance with name 'VI_1' was created which will be monitoring the eth0 interface as a master. The virtual router id of '77' was assigned which acts as the identifier between two load balancers. It is the same for both load balancer nodes. The priority was set as '101', the value in priority helps to elect the master node. Always the node with the higher priority act as a master. In this case, Master node was given the priority of '101' and backup was given '100'. The floating IP address or virtual IP address was assigned as 192.168.1.15. It is the IP address where the client target to get inside the service. The virtual IP can be anything from the same subnet range. Finally, the above created script was tracked and the configuration was saved and the keepalived was restarted using the command *service keepalived restart* as a root user.

Similarly, LbSlave was also configured in a similar way. The configuration was different only in three places. In the slave node, the state was changed from master to backup. Similarly, priority was set to '100' which must be less than the priority id of the master

node. A monitor interface was set to 'Wlan0' to check and test the traffic coming in from the wireless network. In this way, keepalived was configured successfully.

3.5 Mirroring Web Directory with SSH, Cron Job and Rsync

Rsync which stands for remote sync is the utility to back up the files across multiple computers or servers. It only copies the part of the files that have been modified reducing the amount of data copied. In addition, it has ability to delete the files from the mirror which has been deleted from the main node. Rsync also protect the ownership and permission of backup files and directories which need Rsync to configure as a root user. The permission of files and folders are automatically updated once it is modified in the main server. [24]. Rsync works over SSH protocol, so it is necessary the SSH is established between the nodes. Similarly, crontab is a list of scripts which runs on the configured schedule automatically.

In the project, the SSH key pair was shared initially between the nodes which was followed by the installation and configuration of Rsync. After the successful configuration of Rsync, cronjob was created to mirror the files between the nodes in every five minutes. The steps listed below show how the Rsync was configured.

- Initially Rsync was installed on both the web server nodes i.e. Web01 and Web02 by entering the command apt-get *install rsync* as a root user.
- New user *sandeep* was created on Web01 with permission of at least able to read the directory of files in web server. The user *sandeep* was created by inputting the command *useradd -d /home/sandeep -m -s /bin/bash sandeep* & password was given by entering the command *passwd sandeep.* The command inputted initially will also create the valid login shell for *sandeep* which gives ability to access SSH.
- Public/Private key pair was created on mirror i.e. Web02 as a root using the command *mkdir /root/rsync* and *ssh-keygen -t dsa -b 1024 -f /root/rsync/mirror-rsync-key*. After generating the key, it was copied to Web01 using the command *scp /root/rsync/mirror/mirror-rsync-key.pub* sandeep@192.168.1.20:/home/sandeep. This command will copy the SSH key of Web02 to the directory */home/sandeep* of Web01.
- The following command was inserted in Web01 as a *sandeep* to add the contents of *mirror-rsync-key.pub* to */home/sandeep/.ssh/authorized_keys*.

```
mkdir ~/.ssh
chmod 700 ~/.ssh
mv ~/mirror-rsync-key.pub ~/.ssh/
cd ~/.ssh
touch 600 authorized_keys
cat mirror-rsync-key.pub >> authorized_keys
```

- In order to allow only the rsync connection from 192.168.1.21, the following line was added to */homr/sandeep/.ssh /authorized_keys* in Web01.

```
command="home/sandeep/rsync/checkrsync",
from="192.168.1.21", no-port-forwarding,no-X11-for-
warding, no-pty
```

- For the automation of rsync, Cron job was configured in way so that rsync command works in every five seconds. It was done in Web02 by entering the command *crontab -e* and inserting the following line.

```
*/5 * * * * /usr/bin/rsync -azq –delete –ex-
clude=**/stats –exclude=**/error –ex-
clude=**/files/pictures -e "ssh -i /root/rsync-key"
sandeep@192.168.1.20:/var/www/html /var/www
```

  In the command above 'delete option' means that files that are deleted on Web01 also should be deleted on Web02. Similarly, 'exclude command' means that the files and directory listed in exclude should not be mirrored to Web02.

Rsync was successfully configured and automated using SSH and Cron tab. Testing was done by creating different files and directories and monitoring the changes in every five minutes. Also, the permission of synced files and folder was checked and compared with the original.

4 Testing and Result

During the implementation of the project, there were a few problems encountered. During the configuration, every phase of the configuration was tested independently to ensure that it performs as intended. In addition, there was a test done after the final project was ready. Use of 'ping' command was mostly done to check the connection. Similarly, ip-config, service <service name> status, command was also used to see the status. The problem that arise during the project and method of solution is explained below.

As soon as initialization of Pi was done, first test was done by pinging to the DNS server of google and it failed. Connection to the internet was not established which affected the whole project resulting the difficulty in the installation of open source tools which totally rely on internet. Later by researching on the google, it was found that problem was occurred because name server was not resolved. It was solved by adding the IP address of google name servers on the DNS configuration file. In addition, one more problem was occurred during the project. Raspberry Pi nodes were not able to connect to the router with wireless connection. This happened as the router was changed to a new one. In spite of the fact that there was a new router, the wireless access point name and credentials were made the same as the old one. The router had the dual band feature and Raspberry Pi was designed to be get connected with 5 GHz band. Here the blunder was the Wi-Fi dongle used in the Raspberry Pi which supports 2.2 to 2.5 GHz, which made Pi unable to connect to the Wi-Fi network. As soon as the problem was detected, the band was changed from 5 GHz to 2.4 GHz and everything returned normal state.

Web server nodes were tested by browsing the individual's nodes with their correspondent IP address from another machine in the network. After the installation of Rsync between the web server nodes, files were created on master node and checked if it sync with in five minutes on backup node. Also the permissions and size of the files and directory were reviewed. Similarly, Load balancer nodes testing by shutting down one another and checking the IP float by VVRP. The virtual IP was floating according to the availability of load balancer nodes. Health check-up of the load balancer was done by creating the blank file which acted as the key file to report the status of HAProxy load balancer. As a whole, the system was tested by creating multiple web traffic and monitoring the request sent and received back. Load balancer nodes and web server nodes were turned on and off and traffic was monitored.

Finally, the concept of cluster computing was successfully designed and implemented using Raspberry Pi and open-source software to obtain the principle of high availability.

5 Conclusion

The final year project was initiated by following the cluster computing principle to obtain the high availability. It was done in two parts, basic initialization and adding the high availability features. As a result, the goal was achieved and a high available web server was configured. The project was done to minimize the downtime and service interruption with backup nodes at the time of failure.

In the project, Raspberry Pi was used as a mini computing component which is not powerful enough to be used in a production environment. Regardless of the capacity of Raspberry Pi, the project was designed and configured considering the need of the real world. The design provides the flexibility to add or reduce the nodes according to the need. Adding more, the project also has some drawbacks i.e. the network. The project fully relies on the network and network equipment. If the network fails, the whole project will go down. We can solve this problem by adding redundant links to the nodes and adding the backup routes. Similarly, project could have been improved by adding a separate node for storage.

To sum up, this project benefits the readers by providing a information about high availability and cluster computing. After reading this thesis, reader might be motivated towards cluster computing and will be enthusiastic to implement the project in their home or office environment.

References

[1] Microsoft. Designing Web Application, Microsoft [Online]. Available: https://msdn.microsoft.com/en-us/library/ee658099.aspx. Accessed 2017 May 14.

[2] M. Papgelis. Web App Acrhitecture, University of Toronto. [Online]. Available: http://www.cs.toronto.edu/~mashiyat/csc309/Lectures/Web%20App%20Architectures.pdf. Accessed 2017 May 14.

[3] S. Hessinger. Small Buisnees Trends. 2013. [Online]. Available: https://smallbiztrends.com/2013/08/amazon-down-custom-error-page.html. Accessed 23 March 2017.

[4] Microsoft. Understanding Downtime, Microsoft. 20 May 2005. [Online]. Available: https://technet.microsoft.com/en-us/library/aa998704(v=exchg.65).aspx. Accessed 24 March 2017.

[5] E. Marcus and H. Stern. Blueprints for High Availability, Wiley Publishing Inc.. 2003.

[6] E. Marcus. The myth of the nines, TechTarget. August 2003. [Online]. Available: http://searchstorage.techtarget.com/tip/The-myth-of-the-nines. Accessed 27 March 2017.

[7] L. L. Chowdry. Cluster Computing. 20 September 2005. [Online]. Available: https://www.codeproject.com/Articles/11709/Cluster-Computing. Accessed 15 March 2017.

[8] P. Merkey. Beowulf History, [Online]. Available: http://www.beowulf.org/overview/history.html. Accessed 22 March 2017.

[9] A. . B. Luiz, J. Dean and U. Hölzle. Web Search For A Planet: The Google Cluster Architecture, IEEE Computer Society. 2003.

[10] T. Sterling. Beowulf Cluster Computing with Linux, 1st ed., London: The MIT Press. 2002.

[11] Microsoft. Failover Cluster Step-by-Step Guide: Configuring the Quorum in a Failover Cluster. 2011. [Online]. Available: https://technet.microsoft.com/en-us/library/cc770620(d=printer,v=ws.10).aspx. Accessed 15 March 2017.

[12] Microsoft. Microsoft High Availability Overview, Microsoft. 2008.

[13] R. Bansode. What is difference between horizontal & vertical scaling ?, ESDS Software Solution Pvt. Ltd.. 21 December 2013. [Online]. Available: https://www.esds.co.in/blog/what-is-the-difference-between-horizontal-vertical-scaling/#sthash.PMfffJqI.PRtysHzh.dpbs. Accessed 2017 May 14.

[14] M. Schmidt. Raspberry Pi: A Quick-Start Guide, 2nd Edition ed., Dallas, Texas: The Pragmatic Bookshelf. 2014.

[15] W. Harrington. Learning Raspbian, Birmingham: Packt Publishing Ltd. 2015.

[16] The Linux Information Project, Daemon Definition, LINFO. 16 August 2005. [Online]. Available: http://www.linfo.org/daemon.html. Accessed 06 April 2017.

[17] M. Garrels. Bash Guide for Beginners, Garrels BVBA. 2008.

[18] A. K. Dennis. Raspberry Pi Home Automation with Arduino, Birmingham: Packt Publishing Ltd. 2015.

[19] D. J. Barrett, R. E. Silverman and R. G. Byrnes. SSH, the Secure Shell, Sebastopol, CA: O'Reilly Media, Inc. 2005.

[20] Oracle. Oracle Linux Administrator's Guide for Release 6, Oracle Corporation. 2013. [Online]. Available: https://docs.oracle.com/cd/E37670_01/E41138/html/section_hpx_tmb_nr.html. Accessed 6 April 2017.

[21] C. Fidao. Load Balancing with HAProxy. 15 July 2015. [Online]. Available: https://serversforhackers.com/load-balancing-with-haproxy. Accessed 06 April 2017.

[22] W. Tarreau. HAProxy. 13 March 2016. [Online]. Available: https://www.haproxy.org/download/1.4/doc/configuration.txt. Accessed 05 April 2017.

[23] R. V. Elst. Simple Keepalived failover setup on Ubuntu 14.04. 13 June 2014. [Online]. Available: https://raymii.org/s/tutorials/Keepalived-Simple-IP-failover-on-Ubuntu.html. Accessed 07 April 2017.

[24] J. Ellingwood. How To USe Rsync to Sync Local and Remote Directories on a VPS, DigitalOcean. 10 September 2013. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-use-rsync-to-sync-local-and-remote-directories-on-a-vps. Accessed 11 April 2017.