私立東海大學

資訊工程研究所

碩士論文

指導教授：楊朝棟 博士

# 使用虛擬化容錯機制用於改善 Hadoop 檔案系統可靠性

# Improving Availability of Hadoop File System with Virtualization Fault Tolerant Mechanism

研究生：周威利

中華民國 一百年七月

# 東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 周 威 利 所提之論文

使用虛擬化容錯機制改善 Hadoop 檔案系統的可用性

經本委員會審查，符合碩士學位論文標準。

學位考試委員會
召 集 人 楊武 簽章

委 員 時文中

張主力

吳老飞

指 導 教 授 楊朝棟 簽章

中華民國 100 年 6 月 27 日

# 摘要

在大部份的情況下，叢集型使用的架構可分為前端節點與後端節點這樣的部署方式，因此，在這樣子的架構底下我們可以彈性的增加或減少後端節點的數量來應付高度變化的系統資源需求，然而，如何確保前端節點的高度使用性是目前在這樣子的系統架構中所要面臨且必須探討的議題。虛擬化是一種常見的策略以改善現有的計算資源，尤其是在雲計算領域。 Hadoop 是阿帕契(Apache)專案中之一的項目，它的設計規模從一台服務主機到成千上萬的機器，每個地方提供計算和存儲。但是，如何保證穩定性和可靠性將是很好的研究課題。在本研究中，我們使用開放軟體及平台以達到我們的目標，例如 Xen 的管理程序虛擬化技術、OpenNebula 虛擬機的管理工具，等等。經過擴充元件之功能，我們利用其元件提供的服務開發了一種機制，以支持我們的想法，並達成了 Hadoop 的高可用性，我們稱之為虛擬化容錯機制，我們基於這樣設計的方式，來達成我們所要求的高度化的使用性。本文裡我們考慮了一個實際且經常會發生的問題套用在我們的系統上，而結果也證實了其停機時間更是能將之縮短至最小可能，套用此方法的可能性應用方案當然可以不僅止於 Hadoop 的應用，更可以擴展到更多的叢集型系統的領域。

關鍵字：高可用性, 雲端運算, 虛擬化, 虛擬化容錯機制

# Abstract

In most cases, cluster-based architecture can be divided into front-end node (Front-end) and back-end node (Back-end), we can increase or reduce the number of back-end nodes flexibility to face with highly variable system resource requirements. However, how to ensure that front-end node (Front-end) High Availability is an issue needs to be explored and studied. Virtualization is a common strategy for improving the existing computing resources, particularly within cloud computing field. Hadoop, one of Apache projects, it is designed to scale up from single servers to thousands of machines, each offering local computation and storage. However, how to guarantee stability and reliability will be good study topics. In this article, we were using currently open-source based software and platform to hit our goal. For instance, Xen-Hypervisor virtualization technology, OpenNebula virtual machines management tool, etc. After extended component capabilities, we developed a mechanism to support our idea, and reached Hadoop High Availability, which called Virtualization Fault Tolerance (VFT). This thesis we consider a practical problem that occur frequently in our system today, the results also confirmed the downtime time can shorten if failure occurred. In this case, it is not only for the Hadoop applications, but also can be extended to more areas of cluster-based systems.

Keywords: High Availability, Cloud Computing, Virtualization, Virtualization Fault Tolerance (VFT)

# Acknowledgements

第一次踏入東海校園時，隨即被深深的人文及景色所吸引，在隨後的二年時光裡接受了豐富的校園文化薰陶及專業知識訓練外，還有高效能實驗室提供了充足的資源，讓我有機會參與各種研討會及競賽，在東海高效能計算實驗室培養之下，造就出的韌性、態度及知識，以應付未來更多的挑戰，培育了專業的素養與國際觀，在這樣的條件下才有機會在求學期間獲得建國百年自由軟體競賽的銅牌。

要感謝的人很多，首先要感謝指導教授－楊朝棟 博士，在指導期間給予最多耐心且鼓勵，無論是參與研討會前或是參與競賽前的指導，又或是平時的會議討論中給予的意見，過程中的挑戰與學習讓我很受用，很感激楊老師為我們所付出心血。再來，也要感謝口試委員給予的意見及指正讓本篇論文能更加的完善。還要感謝的是學長、同學：龍騰、裕翔、翔耀、政達、冠傑、本加、智霖、 柏翰等實驗室的伙伴們的互相支持與協助，一路走來之下才讓我能有學業完成的一天；過程之中有甜有苦，特別我們都是利用下班之餘或是例假日時間進行學校的課業研究，在有限的時間條下能共同寫作及討論，這樣的革命情感及友情讓我特別感到珍惜。

此外我也要特別感謝我的家人及女友佳郁，沒有妳們的支持與體諒，我將無法順利完成論文，更遑論畢業取得碩士學位。最後，謹以拙作獻給所有關心我的家人、師長與朋友們，願將此喜悅與你們一同分享。

# Tables of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Virtual machine (Virtual Machine) in recent decades, the annual growth rate has significantly improved [1-5]. Whether the vendor's various related products, or is gradually emerging applications in different fields: for example, the CPU instruction set from the most basic dynamic conversion, analog to the popular energy saving, cluster management, or behavior detection and so on. Virtualization technology not only provides significant secondary effects, and even applications in many fields place. In addition to the scope of application of virtualization technology continues to expand, broaden, the virtual machine guest operating system in operation (Guest OS) time also continue to improve efficiency.

Hadoop [6-13] was inspired by Google's MapReduce and Google File System (GFS) [14, 15]papers which provided access to the file systems supported by Hadoop. Hadoop cluster will include a single master and multiple worker nodes. The master node consists of a JobTracker, task tracker, NameNode, and DataNode. The Hadoop Distributed File System (HDFS) uses this when replicating data, to try to keep different copies of the data on different racks. The goal is to reduce the impact of a rack power outage or switch failure so that, even if these events occur, the data may still be readable. However, it even so took a long time to restart the system when failure occurred.

## 1.1 Motivation

To using a new technology could be a big challenge for some people, but in this section, we are not trying to discuss the learning curve, but talk about the reliability and stability issues. Hadoop like others distribution system, it allows you use among back-end resources to operate complex computing or as a huge storage. And front-end charge of metadata link or resource allocation works. Developers could use these feature to achieve services everywhere.

In this thesis, we let Hadoop Namenode running on virtual machine, and developed a high availability mechanism for Namenode. HDFS filesystem instance requires one unique server, the name node. This is a single point of failure for an HDFS installation. If the name node goes down, the filesystem is offline. When it comes back up, the name node must replay all outstanding operations. This replay process can take over half an hour for a big cluster. The filesystem includes what is called a Secondary Namenode, which misleads some people into thinking that when the Primary Namenode goes offline, the Secondary Namenode takes over. In fact, the Secondary Namenode regularly connects with the Primary Namenode and builds snapshots of the Primary NameNode's directory information, which is then saved to local/remote directories. These checkpointed images can be used to restart a failed Primary Namenode without having to replay the entire journal of filesystem actions, the edit log to create an up-to-date directory structure.

Various challenges are faced while developing a distributed application [3, 16-21]. The first problem to solve is hardware failure: as soon as we start using many pieces of hardware, the chance that one will fail is fairly high. The second problem is that most analysis tasks need to be able to combine the data in some way; data read from one disk may need to be combined with the data from any of the other disks. However, the HDFS and MapReduce the original Hadoop kernels, are already fixed this problem. HDFS allows replication redundant

copies of data are kept by the system so that in the event of failure, there is another copy available. This is mostly like RAID works. And the MapReduce provides a programming model that abstracts the problems from disk read and writes to transform into a computation over sets of keys and values.

However, Hadoop does not support automatic recovery in the case of a NameNode failure. This is a well-known and recognized single point of failure in Hadoop. In Hadoop Official site mentioned about this [6]: if the NameNode machine fails, manual intervention is necessary. Currently, automatic restart and failover of the NameNode software to another machine is not supported. If the NameNode single-point-of-failure is established, then trying to solve this problem is the goal of this thesis.

## 1.2 Contributions

Hadoop infrastructure has become a critical part of day-to-day business operations. As such, it was important for us to find a way to resolve the single-point-of-failure issue that surrounds the master node processes, namely the NameNode and JobTracker. While it was easy for us to follow the best practice of offloading the secondary NameNode data to an NFS mount to protect metadata, ensuring that the processes were constantly available for job execution and data retrieval were of greater importance. We've leveraged some existing, well tested components that are available and commonly used in Linux systems today. Our solution primarily makes use of DRBD [1] from LINBIT and Heartbeat from the Linux-High Availability (HA) project which we called Virtualization Fault Tolerance (VFT). The natural combination of these projects provides us with a reliable and highly available solution, which addresses limitations that currently exist.

Virtualization is being used as a solution not only to provide service flexibility, but also

to consolidate server workloads and improve server utilization. A virtualized based system can be dynamically adapted to the client demands by deploying new virtual nodes when the demand increases, and powering off and consolidating virtual nodes during periods of low demand. In this thesis, we employ the virtual machine management tool, OpenNebula [22-24], to manage virtual machines and combine others open source resources to achieve Hadoop Namenode high availability goal.

## 1.3 Thesis Organization

This thesis is organized as follows. First, we start the background and related works of this thesis in chapter 2. Chapter 3 describes the system implementation and shows how we design the VFT mechanism, and it also shows the interface of our virtual machine management tool. In chapter 4, we design some scenarios to prove our system and mechanism. Finally, chapter 5 outlines main conclusions and the future works.

# Chapter 2

# Background Review and Related Work

## 2.1 Apache Project: HADOOP

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project. Hadoop is best known for MapReduce and its distributed filesystem (HDFS, renamed from NDFS), the term is also used for a family of related projects that fall under the infrastructure for distributed computing and large-scale data processing. The Hadoop projects that are covered projects as below lists:

a. ***Common*** : a set of components and interfaces for distributed file systems and general I/O (serialization, Java RPC, persistent data structures).

b. ***Avro:*** a serialization system for efficient, cross-language RPC, and persistent data storage.

c. ***MapReduce:*** a distributed data processing model and execution environment that runs on large clusters of commodity machines.

d. ***HDFS***: a distributed filesystem that runs on large clusters of commodity machines.

e. ***Pig:*** a data flow language and execution environment for exploring very large datasets.

f. Pig runs on HDFS and MapReduce clusters.

g. ***Hive:*** a distributed data warehouse. Hive manages data stored in HDFS and provides a query language based on SQL (and which is translated by the runtime engine to

MapReduce jobs) for querying the data.

h. ***HBase:*** a distributed, column-oriented database. HBase uses HDFS for its underlying storage, and supports both batch-style computations using MapReduce and point queries (random reads).

i. ***ZooKeeper:*** a distributed, highly available coordination service. It provides primitives such as distributed locks that can be used for building distributed applications.

j. ***Sqoop:*** a tool for efficiently moving data between relational databases and HDFS.

## 2.2 High Availability

High Availability[25] means "A system design approach and associated service implementation that ensures a prearranged level of operational performance will be met during a contractual measurement period". We will focus this on cloud configurations that remove as many single points of failure as possible and that are inherently designed with a specific focus on operational continuity, redundancy, and fail-over capability

High Availability can be achieved at many different levels, including the **application level, infrastructure level**, **data center level**, and **geographic redundancy level**. We will focus on the infrastructure level in this thesis. All system outages fall into two major categories [19, 26]:

**Unplanned System Outages**: unplanned outages are the result of uncontrollable random system failures associated with faults occurring within hardware or software components. Unplanned outages are the most costly, with the highest gains being achieved when steps are taken to avoid this type of outage.

**Planned System Outage**: a planned outage should be scheduled to have a minimum availability impact on a system. Planned outages is the result of maintenance events revolving

around repair, backup, or upgrade operations. Repairs are intended to remove faulty components and restore a system to a functional state. Backups are intended to preserve critical data on a magnetic storage medium (disk or tape) to avoid the loss of data when a production system experiences a main disk storage failure. Upgrades are implemented to replace the current hardware or software with newer (or enhanced) versions.

Do we need high availability for our business? It's all depending on your system level as the previous definition. But in our daily lives some applications' downtime will create a significant amount of angst for users. While Facebook, Gmail, or AT&T experience outages, these events will receive national and often international attention. Not only can downtime in your products turn into a disastrous PR nightmare, but more importantly, it can also seriously tarnish the loyalty of your customer base that depends on you for their financial livelihood. Regardless of the size of your organization, if downtime in your internal infrastructure or core product offerings negatively impacts your bottom line, you are a perfect candidate for exploring cloud HA. High Availability can still be a tricky and expensive proposition in dedicated environments.

In HA field, Floyd Piedad [26] also presented availability level and measurement. In same field they all indicated that IT must understand the level of availability users require, and users must understand the costs of achieving these targets. Of all availability levels, continuous availability is the most challenging and expensive to provide. In our work, we are toward this topic and trying to make it available.

## 2.3 Fault Tolerance Technology

In this thesis, we consider DRBD with Heartbeat is a good solution fault tolerance technology. DRBD is a short name of "Distributed Replicated Block Device" (DRBD) is a software-based,

shared-nothing, replicated storage solution mirroring the content of block devices (hard disks, partitions, logical volumes, etc.) between servers.

**DRBD mirrors data**: in real time: replication occurs continuously, while applications modify the data on the device.

**Transparently**: the applications that store their data on the mirrored device are oblivious of the fact that the data is, in fact, stored on several computers.

**Synchronously or asynchronously**: With synchronous mirroring, a writing application is notified of writing completion only after the writing has been carried out on both computer systems. Asynchronous mirroring means the writing application is notified of writing completion when the writing has completed locally, but before the writing has propagated to the peer system.

DRBD technology is designed as a block device building block to form a high-availability (HA) cluster. This is done by mirroring a whole block device via a specified network. DRBD technology can be understood as a network RAID-1. The bottom part of this illustration shows a cluster where the left node is currently activated, i.e., the service's IP address that the client machines are talking to which is currently on the left node. The service, including its IP address, can be migrated to the other node at any time, either due to a failure of the active node or as an administrative action. The lower part of the illustration shows a degraded cluster. In HA speak the migration of a service is called failover. The reverse process is called failback and when the migration is triggered by an administrator it is called switchover [27]. In Figure 2-1, it displays entire DRBD architecture.

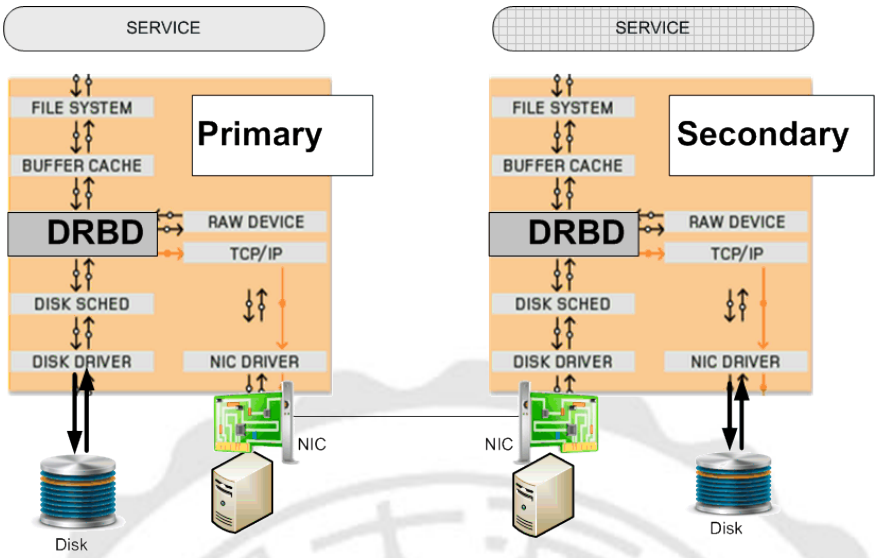


Figure 2-1. DRBD Architecture

DRBD's core functionality is implemented by way of a Linux kernel module. In additional, DRBD constitutes a driver for a virtual block device, so DRBD is situated “right near the bottom” of a system's I/O stack. Because of this, DRBD is extremely flexible and versatile, which makes it a replication solution suitable for adding high availability to just about any application.

**Heartbeat**[28] is a daemon that provides cluster infrastructure (communication and membership) services to its clients. This allows clients to know about the presence (or disappearance!) of peer processes on other machines and to easily exchange messages with them [30]. As Figure 2-2 shown, DRBD with Heartbeat is a fault tolerance solution in Linux based OS. And this solution is also very important in this thesis.

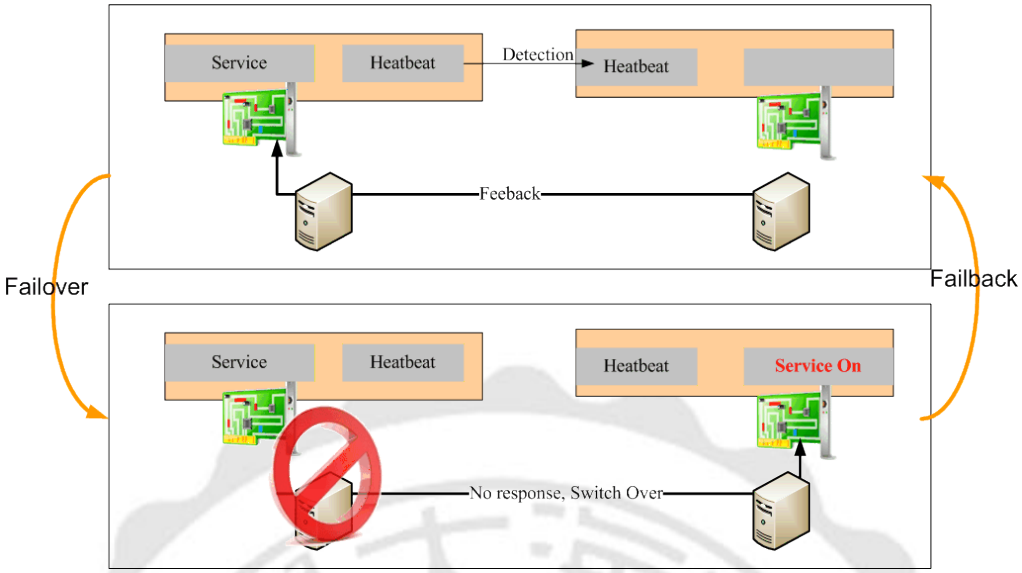


Figure 2-2. A Life Cycle of Heartbeat activities

## 2.4 Virtualization Technologies

Virtualization technology [1, 5, 18, 29-32] is an interesting solution to implement cluster-based server and to overcome these problems. Cluster nodes can be virtualized using some virtualization platform (Xen, KVM, VMWare, etc.) and can be managed by an efficient virtual machine manager that could incorporate a provisioning model for dynamically deploying new virtual cluster nodes when the user demand increases, and consolidate virtual nodes when the demand decreases. Virtualization lets you run multiple virtual machines on a single physical machine, with each virtual machine sharing the resources of that one physical computer across multiple environments.

Virtualization is simply the logical separation of the request for some service from the physical resources that actually provide that service. In practical terms, virtualization provides the ability to run applications, operating systems, or system services in a logically distinct system environment that is independent of a specific physical computer system. Obviously, all

of these have to be running on a certain computer system at any given time, but virtualization provide a level of logical abstraction that liberates applications, system services, and even the operating system that supports them from being tied to a specific piece of hardware. Virtualization, focusing on logical operating environments rather than physical ones, makes applications, services, and instances of an operating system portable across different physical computer systems. Virtualization can execute applications under many operating systems, manage IT more efficiently, and a lot resource of computing with other computers .

In general, most virtualization strategies fall into one of three major categories [1]: **Full Virtualization**: Also sometimes called hardware emulation. In this case an unmodified operating system is run using a hypervisor to trap and safely translate/execute privileged instructions on-the-fly. Because trapping the privileged instructions can lead to significant performance penalties, novel strategies are used to aggregate multiple instructions and translate them together. Other enhancements, such as binary translation, can further improve performance by reducing the need to translate these instructions in the future.

**Para-virtualization**: Like full virtualization, para-virtualization also uses a hypervisor, and also uses the term virtual machine to refer to its virtualized operating systems. However, unlike full virtualization, para-virtualization requires changes to the virtualized operating system. This allows the VM to coordinate with the hypervisor, reducing the use of the privileged instructions that are typically responsible for the major performance penalties in full virtualization. The advantage is that para-virtualized virtual machines typically outperform fully virtualized virtual machines. The disadvantage, however, is the need to modify the para-virtualized virtual machine/operating system to be hypervisor-aware. This has implications for operating systems without available source code.

**Operating System-level Virtualization:** The most intrusive form of virtualization is operating system level virtualization. Unlike both para-virtualization and full virtualization,

operating system-level virtualization does not rely on a hypervisor. Instead, the operating system is modified to securely isolate multiple instances of an operating system within a single host machine. The guest operating system instances are often referred to as virtual private servers (VPS). The advantage to operating system-level virtualization lies mainly in performance. No hypervisor/instruction trapping is necessary. This typically results in system performance of near-native speeds. The primary disadvantage is that all VPS instances share a single kernel. Thus, if the kernel crashes or is compromised, all VPS instances are compromised. However, the advantage to having a single kernel instance is that fewer resources are consumed due to the operating system overhead of multiple kernels.

For the remainder of this thesis, we choose “Xen Hypervisors” to be our virtualization technology platform. It is the basic abstraction layer of software that sits directly on the hardware below any operating systems. It is responsible for CPU scheduling and memory partitioning of the various virtual machines running on the hardware device. The hypervisor not only abstracts the hardware for the virtual machines but also controls the execution of virtual machines as they share the common processing environment. It has no knowledge of networking, external storage devices, video, or any other common I/O functions found on a computing system.

In same research, P. Barham [29] presented Xen, an x86 virtual machine monitor which allows multiple commodity operating systems to share conventional hardware in a safe and resource managed fashion, but without sacrificing either performance or functionality. This is achieved by providing an idealized virtual machine abstraction to which operating systems such as Linux, BSD and Windows XP, can be ported with a minimal effort

## 2.5 Virtual Machine Management

A key component in this scenario is the virtual machine (VM) management system. A VM manager provides a centralized platform for efficient and automatic deployment, control, and monitoring of VMs on a distributed pool of physical resources. Usually, these VM managers also offer high availability capabilities and scheduling policies [33]. Eucalyptus, OpenNebula and Nimbus [22-24, 34] are three major open-source cloud-computing software platforms. The overall function of these systems is to manage the provisioning of virtual machines for a cloud providing infrastructure-as-a-service. These various open-source projects provide an important alternative for those who do not wish to use a commercially provided cloud. In this thesis we employ OpenNebula to be the research object.

The OpenNebula is a virtual infrastructure engine that enables the dynamic deployment and re-allocation of virtual machines in a pool of physical resources. The OpenNebula system extends the benefits of virtualization platforms from a single physical resource to a pool of resources, decoupling the server, not only from the physical infrastructure but also from the physical location. The OpenNebula contains one frontend and multiple backend. The front-end provides users with access interfaces and management functions. The back-ends are installed on Xen servers, where Xen hypervisors are started and virtual machines could be backed. Communications between frontend and backend employ SSH. The OpenNebula gives users a single access point to deploy virtual machines on a locally distributed infrastructure.

OpenNebula orchestrates storage, network, virtualization, monitoring, and security technologies to enable the dynamic placement of multi-tier services (groups of interconnected virtual machines) on distributed infrastructures, combining both data center resources and remote cloud resources, according to allocation policies. In Figure 2-3, the OpenNebula internal architecture can be divided into three layers.

a. **Tools**, management tools developed using the interfaces provided by the OpenNebula Core.

b. **Core**, the main virtual machine, storage, virtual network and host management components.

c. **Drivers**, it is to plug-in different virtualization, storage and monitoring technologies and Cloud services into the core.

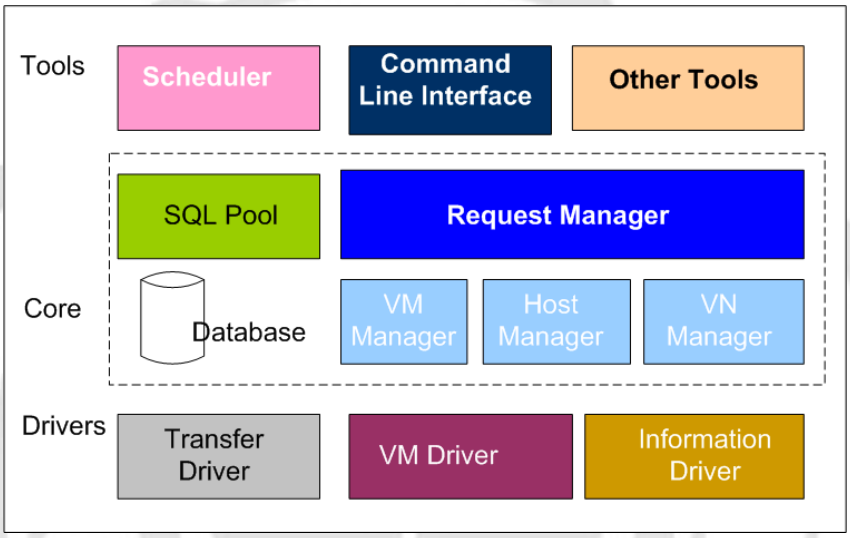


Figure 2-3. OpenNebula Internal Architecture

In previous works we build virtual machines on OpenNebula and implemented Web-based management tool. Thus, the system administrator can be easy to monitor and manage the entire OpenNebula System on our project. OpenNebula is composed of three main components: (1)the *OpenNebula Core* is a centralized component that manages the life cycle of a VM by performing basic VM operations, and also provides a basic management and monitoring interface for the physical hosts (2) the *Capacity Manager* governs the functionality provided by the OpenNebula core. The capacity manager adjusts the placement of VMs based on a set of pre-defined policies (3) *Virtualizer Access Drivers*. In order to provide an abstraction of the underlying virtualization layer, OpenNebula uses pluggable drivers that expose the basic functionality of the hypervisor [2].

## 2.6 Dynamic Resource Allocation

In our previous paper “A Dynamic Resource Allocation Model for Virtual Machine Management on Cloud” published Dynamic Resource Allocation (DRA) algorithm, which has a detail description of DRA [25], and it is one of the key components of this thesis basis. In this work, we focus on enhance Hadoop HA architecture problem, therefore DRA is not described in detail in this thesis; if you are interested in DRA, please refer to “A Dynamic Resource Allocation Model for Virtual Machine Management on Clusters” article. However, the purpose of DRA is to reach the best balance between each physical machine. To avoid computing resources centralized on some specify physical machines, how to balance the resources is most important issue. To achieve the maximum efficiency the resource must be evenly distributed.

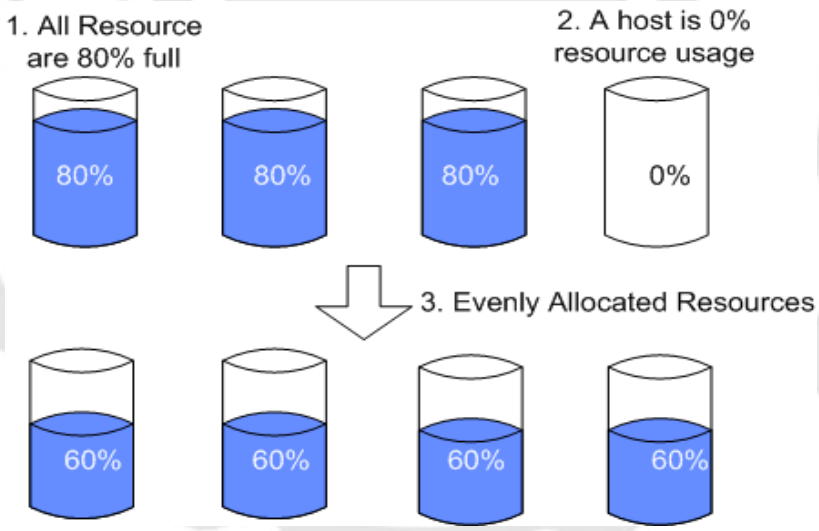


Figure 2-4. Dynamic Resource Allocation

## 2.7 Related Works

In this field, there still have another choice to achieve fault tolerance which is OpenVZ[35], it container-based virtualization for Linux. OpenVZ creates multiple secure, isolated containers

on a single physical server enabling better server utilization and ensuring that applications do not conflict. And in same research, J. Walters and V. Chaudhary [18], proposed “A Fault-Tolerant Strategy for Virtualized HPC Clusters”, using both checkpointing and replication in order to ensure the lowest possible checkpointing overhead. However, they still have some open issues which are how to integration checkpointing and fault-tolerance system into common cluster batch schedulers. But they still provide us a nice practice to handle fault tolerance for virtualization on single site.

G. Vallee [20] proposed a frame to solve fault tolerance issue. Such a framework enables the implementation of various fault tolerance policies, including policies presented in the literature that were not validated by experimentation; therefore G. Vallee presented framework, coupled with their fault tolerance simulator, provides a complete solution for the study of proactive fault tolerance policies. The framework prototype currently provides a single policy based on Xen VM migration but new policies are currently under development in their work. Aim to this point, Xen VM migration issue has been overcome under our framework. The reason is kind of framework need be managed via VM management tool, such as OpenNebula [36].

Regarding to Fault Tolerance mechanism on Hadoop, a good solution has been presented by Cloudera [11]. Cloudera is focus on provide various Hadoop solution. In Sep.2009, Christophe Bisciglia presented an article of "Hadoop HA Configuration". It was implemented Headbeat and DRBD to enhance Hadoop HA, and we extended it on visualization today.

H. Zhong et al. [37] proposed an optimized scheduling algorithm to achieve the optimization or sub-optimization for cloud scheduling problems. In same research, the authors investigated the possibility to allocate the Virtual Machines (VMs) in a flexible way to permit the maximum usage of physical resources. Author used an Improved Genetic Algorithm (IGA) for the automated scheduling policy. The IGA used the shortest genes and

introduces the idea of Dividend Policy in Economics to select an optimal or suboptimal allocation for the VMs requests. This thesis inspired us to find out how to get an optimized algorithm to hit our goal.

Our paper focuses VMs running on physical machines and use DRA technology to implementation virtualization fault tolerance.

# Chapter 3

# System Implementation

In this chapter, we introduce our system architecture and how we composed those components. Of course, the OpenNebula plays a key role in the entire system. The most advantage which is Live Migration function, compare to other virtualization management tools, Live Migration is all they lacked. Since OpenNebula proposed this unique function, we might think another possibility to enhance system high availability, which is combined with DRBD and Heart Beat. In order to achieve this goal, we made following systems and experiments. For more detail, please see below sections.

## 3.1. System Overview

The entire system is according to official OpenNebula manual. The OpenNebula core orchestrates three different management areas: image and storage technologies (that is, virtual appliance tools or distributed file systems) for preparing disk images for VMs, the network fabric (such as Dynamic Host Configuration Protocol servers, firewalls, or switches) for providing VMs with a virtual network environment, and the underlying hypervisors for creating and controlling VMs. The core performs specific storage, network, or virtualization operations through pluggable drivers. Thus, OpenNebula isn't tied to any specific environment, providing a uniform management layer regardless of the underlying infrastructure.

As the shown illustration in Figure 3-1, it is over view of system architecture. According to the previous works, we build a cluster system with OpenNebula and also provide a web interface to manage virtual machines and physical machine. Our cluster system was built up with four homogeneous computers; the hardware of these computers is equipped with Intel i7 CPU 2.8 GHz, four gigabytes memory, 500 gigabytes disk, Debian operating system, and the network connected to a gigabit switch.

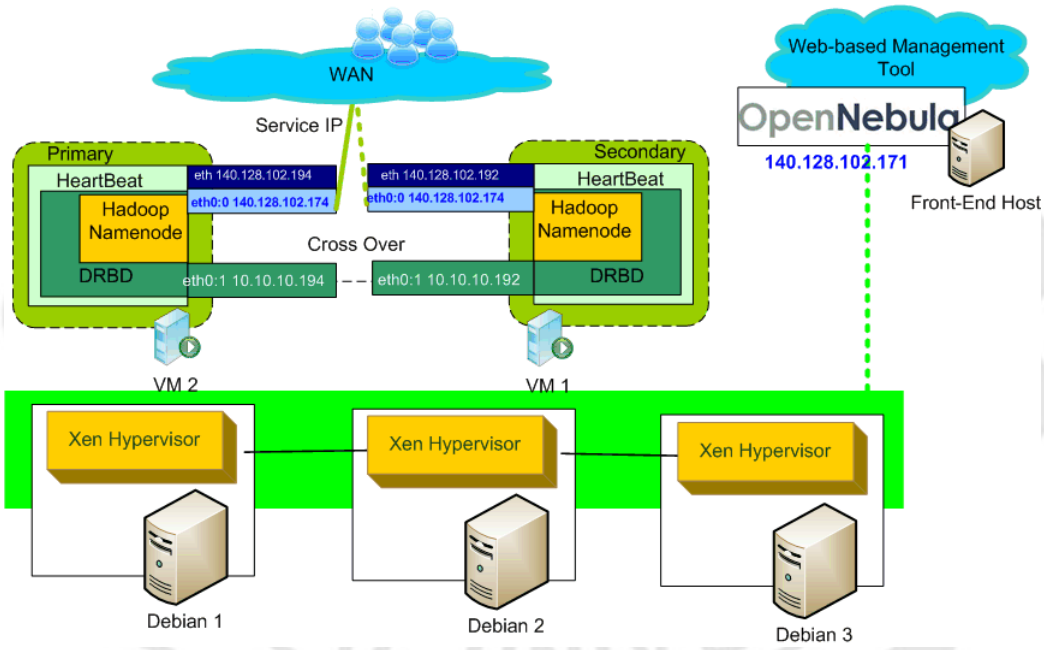


Figure 3-1. System Overview

Follow bottom to top, the infrastructure, Hosts, means physical machines. And Xen Hypervisor is one of virtualization technology suit for Linux series OSs. And follow up are two VMs, VM2 is primary node, and VM1 is secondary node. Assume Hadoop Namenode is built on VM1 as primary node; the VM2 is the slave node of VM1. Under a Heartbeat + DRBD mechanism, you can see we use 5 IP to deploy on this system, one pair is for Cross Over and the other pair is for identify the primary and secondary, the last one is for the service

usage. Finally, a key component OpenNebula is on the top layer, it is the key of entire scenario, this component provides a centralized platform for efficient and automatic deployment, control, and monitoring of VMs on a distributed pool of physical hosts. And we also compose a web interface management tool via DRA and OpenNebula's components to manage virtual machines.

### 3.1.1. Network Configuration

Due to limitation physical IP address, we build a private network environment in our laboratory. Before the HA mechanism was active, some works need be done before that. First, we need to set the IPs on both virtual machines. The IP 192.168.123.210 means Service IP, it is controlled by Heartbeat, and it is using to provide service for users. In this scenario, we assume VM2 is primary node (Table 3-1) and VM1 is secondary (Table 3-2), you can also refer to Figure 3-2.

Table 3-1 VM2 - Primary Node network setting

| **IP Setting** | **Description** |
|---|---|
| eth0 192.168.123.212 | For identify this machine |
| eth0:0 192.168.123.210 | Service IP, Control By Heartbeat to provide services for outside users |
| eth1 10.1.1.211 | For data transfer control by DRBD |

Table 3-2 VM1 - Secondary Node network setting

| **IP Setting** | **Description** |
|---|---|
| eth0 192.168.123.212 | For identify this machine |
| eth0:0 192.168.123.210 | Service IP, control by Heartbeat, disabled when this machine is secondary node |
| eth1 10.1.1.212 | For data transfer control by DRBD |

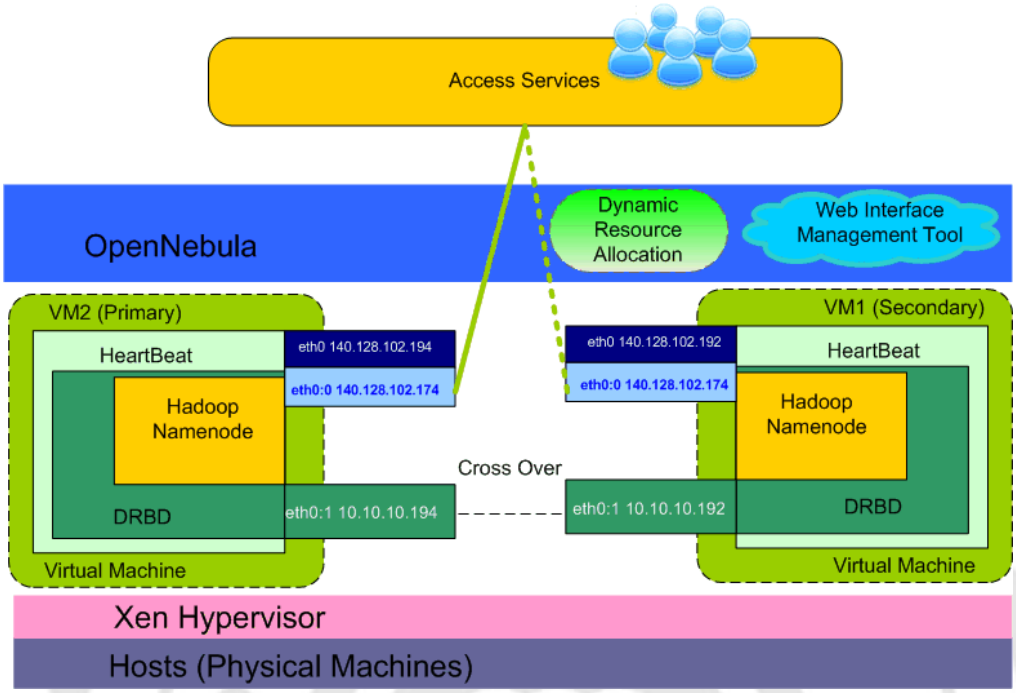


Figure 3-2. Primary and Secondary Nodes Networking Configuration

### 3.1.2. DRBD Configuration

After downloaded the DRBD package and installed it complete [30], then, we could start to set DRBD config file in both two nodes, gave the same setting as shown Figure 3-7. Part of drbd.conf Content in /etc/drbd.conf. For the reminder, it is needed consistency setting in both primary and secondary.

```
global {
   usage-count yes;
}

common {
   syncer { rate    30M; }
}

resource r0 {
   protocol C;
   startup {
```

```
    wfc-timeout 0;
    degr-wfc-timeout 120;
  }

  disk {
    on-io-error     detach;
    # no-disk-flushes;
    # no-md-flushes
    # size 1G;
   }

  net {
   }

on debian-ha1 {               #VM 2
    device      /dev/drbd0;
    disk        /dev/sdb1;
    address     10.1.1.211:7789;
    meta-disk internal;
  }

on debian-ha2 {               #VM 2
    device      /dev/drbd0;
    disk        /dev/sdb1;
    address     10.1.1.212:7789;
    meta-disk internal;
  }
}
```

Figure 3-3. Part of drbd.conf Content

To check the DRBD state with below commands:

#cat /proc/drbd

```
version: 8.0.14 (api:86/proto:86)
GIT-hash: bb447522fc9a87d0069b7e14f0234911ebdab0f7 build by phil@fat-tyre, 2008-11-12 16:40:33
 0: cs:Connected st:Primary/Secondary ds:UpToDate/Diskless C r---
    ns:49 nr:8 dw:12 dr:106 al:1 bm:2 lo:0 pe:0 ua:0 ap:0
        resync: used:0/61 hits:0 misses:0 starving:0 dirty:0 changed:0
        act_log: used:0/127 hits:2 misses:1 starving:0 dirty:0 changed:1
```

or

#drbdadm state r0

```
Primary/Secondary
```

Figure 3-4. Check DRBD State

### 3.1.3. Heartbeat Configuration

There are many options available for the Heartbeat configuration. In this section, we attempt to show our methods. There are there main files that we edit to configure the Heartbeat package:

a. /etc/ha.d/authkeys

b. /etc/ha.d/ha.cf /

c. /etc/ha.d/haresources

First, authkeys should also be the same on both servers. Remember to change the permission as following introduction.

```
chmod 0600 /etc/ha.d/authkeys      #remember to revise the permission
```

Figure 3-5. Revise the authkeys access permission

Second, ha.cf, defines the general settings of the cluster. Our example:

```
## start of ha.cf
logfile /var/log/ha-log
logfacility       local0

keepalive 2             #Detection period
warntime 5
deadtime 20
initdead 120
#hopfudge 1

udpport 694             #Using UDP 694
auto_failback off       #if failback, resume to master
#baud 19200
bcast eth1              #using eth1, to be the heartbeat network card
ucast eth0 192.168.123.211
ucast eth1 10.1.1.211

node debian-ha1      #Node 1, Server Name
node debian-ha2      #Node 2, Server Name

ping 192.168.123.254         #Ping our Gateway, check heart self

respawn hacluster /usr/lib/heartbeat/ipfail
apiauth ipfail gid=haclient uid=hacluster
## end of ha.cf
```

Figure 3-6. Part of ha.cf Content

Finally, the last file, haresources, defines all cluster resources that will fail over from one node to the next. The resources include the Service IP address of the cluster, the DRBD resource “r0” (from /etc/drbd.conf), the file system mount, and the three Hadoop master node initiation scripts that are invoked with the “start” parameter upon failover.

```
debian-ha1 \
192.168.123.210/24 \
drbddisk::r0 Filesystem::/dev/drbd0::/drbd::ext3::noatime \

#initial the services you wanted
hadoop-namenode
```

Figure 3-7. Part of drbd.conf Content

## 3.2. Virtualization Fault Tolerant Methodology

Our approach for the virtual machine's management is an efficient mechanism to reach high available under limited resources. Apart from this, how to research fault-tolerant on virtualization machines and then raise reliability is the topic we want to solve in this thesis. In order to provide continuous availability for applications in the event of server failures a detection methodology is necessary in this thesis.

The Virtualization Fault Tolerance (VFT) has three main phases: ***virtual machine migration policy***, ***information gathering***, and ***keep services always available***. The workflow can be described as follows the illustration (Figure 3-10). However, there is a constraint of this methodology, which is the physical host number must be bigger than three. It is the base requirement to achieve VFT methodology. The coming section will explain this reason.

***Virtual Machine Migration Policy*:** it stands for enabled DRA to make sure the entire distribution virtualization cluster under a best performance.

***Information Gathering*:** this phase is presented that we have a detection mechanism to

retrieve all Hosts and check Hosts is alive or not. We detect the hosts' state with Ping command every five minutes by running Linux schedule via "crontab". In Figure 3-8, its shows how we detect the server state with a Ping command with Shell Script.

```
#!/bin/bash
#spilt hosts with whitespace
HOSTS="host1 hots2"
for LOOP in $HOSTS
do
   if ! ping -c 3 $LOOP > /dev/null 2>&1; then
      echo "Warning:The host $LOOP is unavaiable now! " >> error.log
   fi
done

if [ -f error.log ]; then
   #send mail to myself
   mail -s "Warning:Host is off-line" nagage@gmail.com < error.log
   rm -f error.log
   #star to next phase you. . .
fi
```

Figure 3-8. Collection Hosts State Script

***Keep Service Always Available*:** assume VM m is under Heartbeat + DRBD mechanism and Host n is unavailable physical machine. Once the Host n is shutting down, if VM *m* is the secondary node, then it will move to on-line Host and boot automatically. If VM *m* is the primary node then secondary node will replace the VM *m* to primary node immediately. Next pre-primary node will boot on available host/hots and become secondary. In OpenNebula, command ***onevm*** is to submit, control and monitor virtual machines (Figure 3-9). This helps us control dead VM to deploy on others available physical host.

**onevm livemigrate** <vm_id> <host_id>

or

**onevm migrate** <vm_id> <host_id>

Figure 3-9. Lifemigrate / migrate with OpenNebula command line

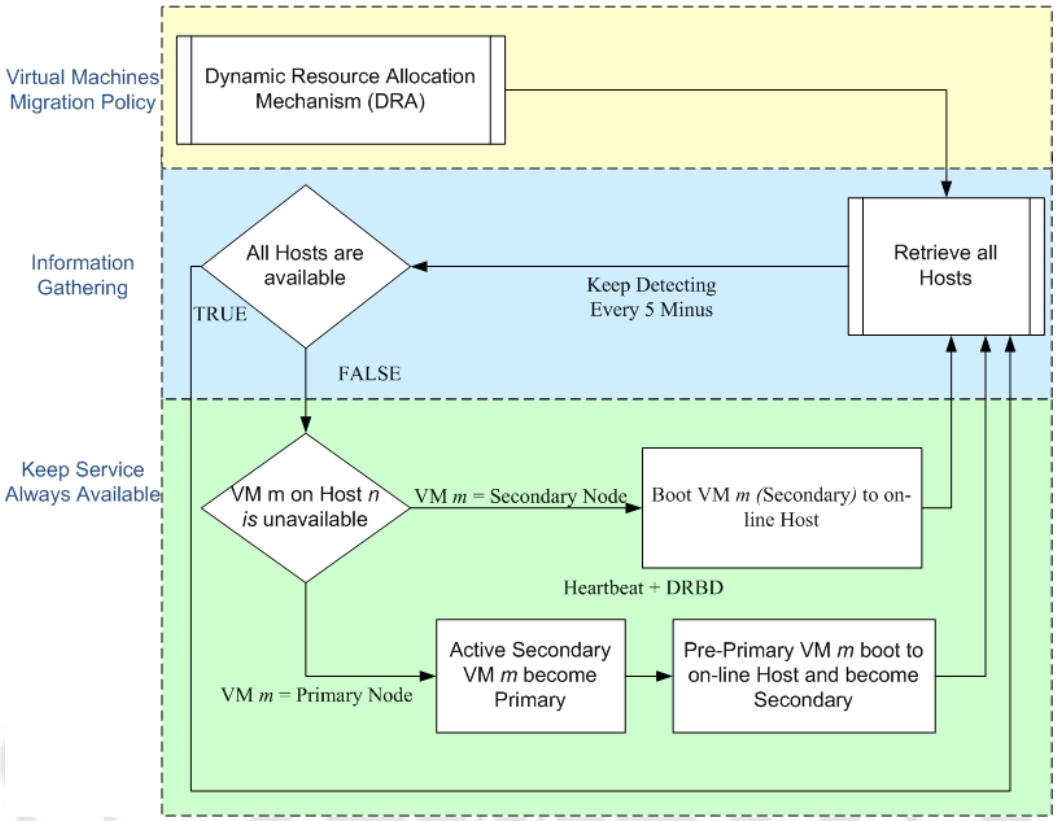


Figure 3-10. Virtualization Fault Tolerance Flow

This flow was made as one of the schedule programs and deployed on front-end. It is reasonable to enhance this function on front-end of OpenNebula, because the OpenNebula control all the VMs operation. There was an example could explain under single-failure event triggered our VFT approach as Figure 3-11.

First one Host *A* was shutting down by unexpected matter, in few minutes later the front-end detected it and also triggered VFT, next, the secondary node VM *2* became primary and handover all services from pre-primary, that we called FAILOVER. Finally, VM *1* booted on Host *C* automatically and became the secondary node, that we called FAILBACK.

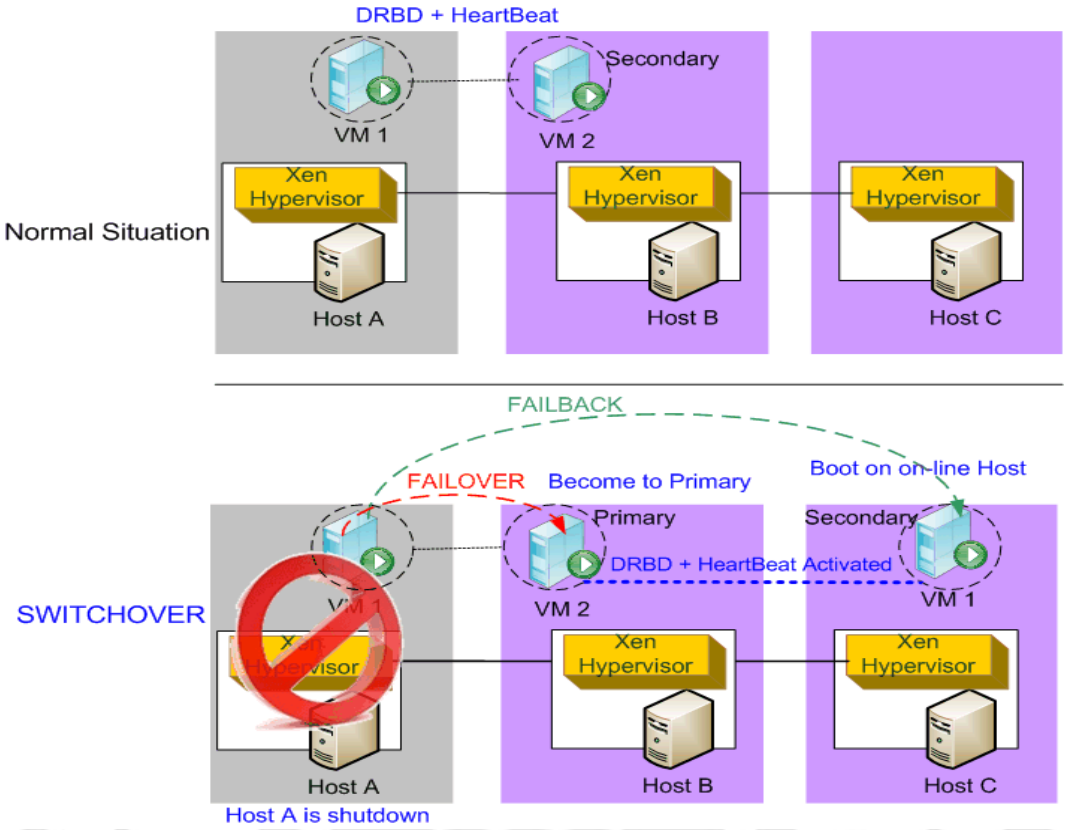


Figure 3-11. How to Trigger VFT

## 3.3. System Interface

As mentioned above, we build a web interface to manage the virtual machines, in following introduction, we would not list all the functions of it, but we focus on this thesis implantation. Figure 3-12, it shows the system authorization, through the core of the web-based management tool, which provided a basic protection for the system. Via this website can control and manage physical machine and VM life-cycle.

RRDtool [38] is the Open Source industry standard, high performance data logging and graphing system for time series data. We use it to create our system log monitor. Resource monitor as shown Figure 3-13, it plays a real important role of entire system. It split into two categories, one is CPU usage percentage and the other one is memory usage state.

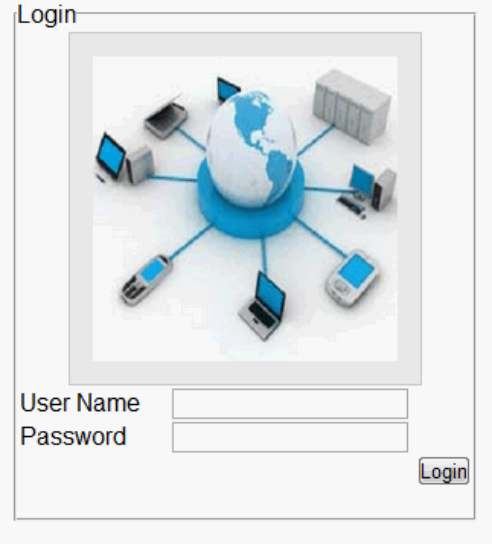


Figure 3-12. System Authorization

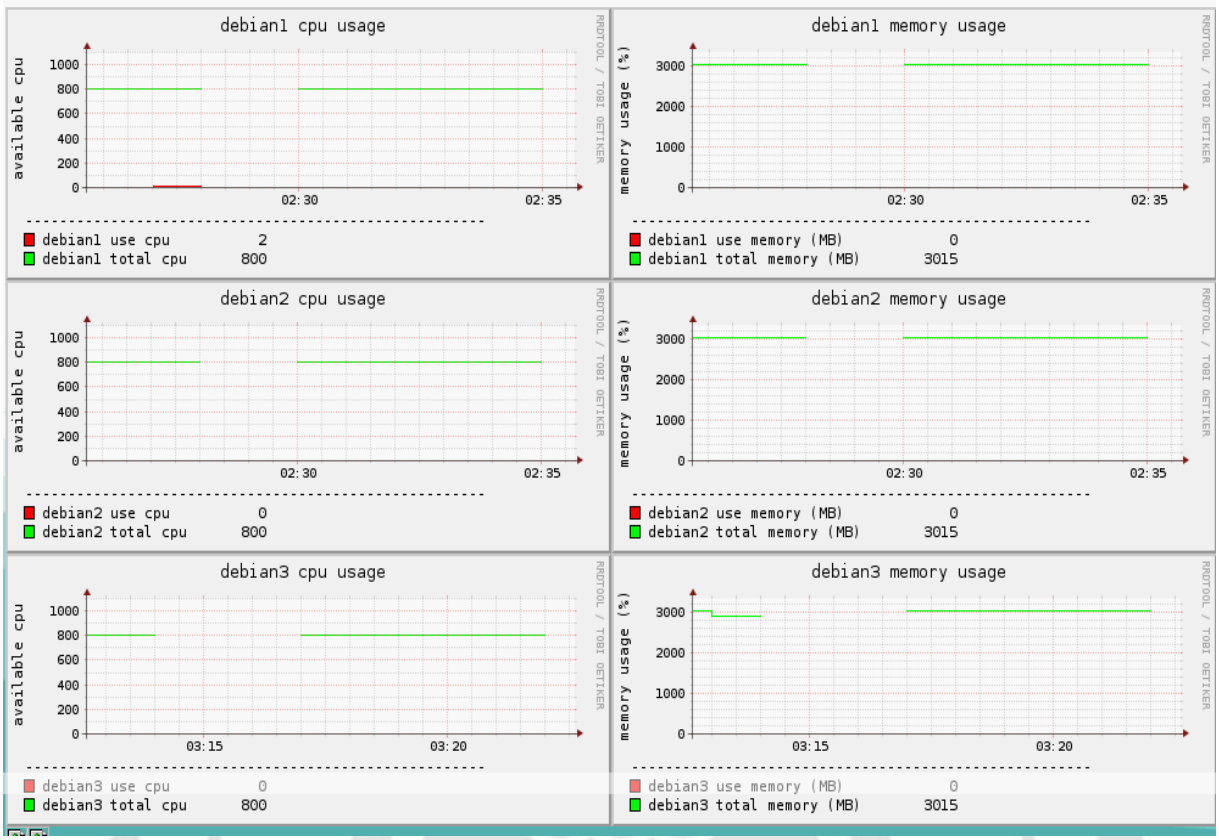


Figure 3-13. Resource Monitor

And then we also provided a quick HA setting as shown Figure 3-14. After HA is enabled, VFT mechanism will be active automatically.

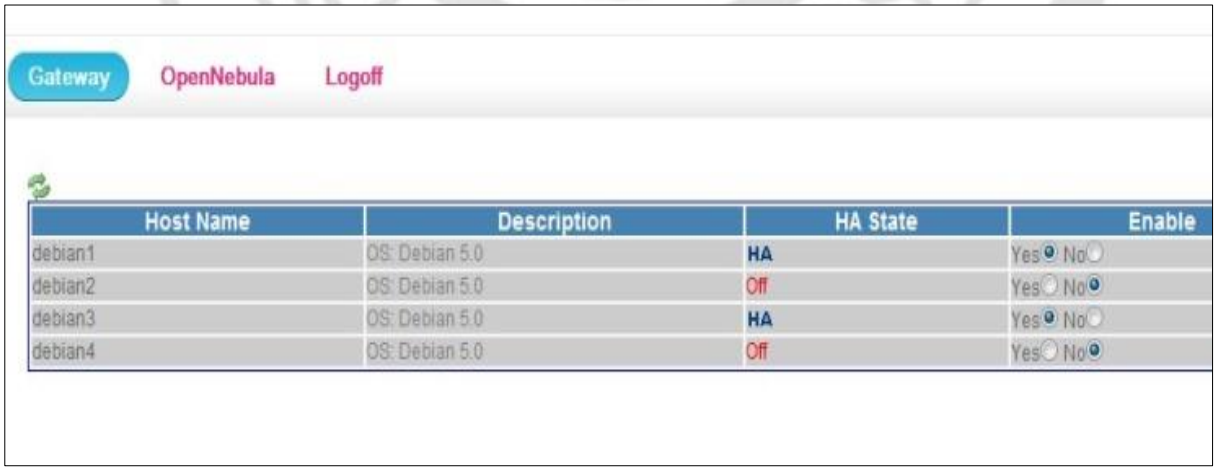


Figure 3-14. High Availability Settings

# Chapter 4

# Experimental Environment and Results

## 4.1. Experimental Environment

In our experimental environment each server has same specification. We give a table list as shown Table 4-1 it descripted our servers CPU, Memory and storages capabilities. And we also measured the basic capability of its performance with known benchmark. Table 4-1 Lab Server Hardware Specification shows the server's hardware specification. In the next experiments, we will via Apache JMeter and HPCC to complete our experiment's data.

Table 4-1 Lab Server Hardware Specification

| No | Model | Cores | CPU MHz | Disk (Giga) | Memory (Giga) | Comments |
|---|---|---|---|---|---|---|
| 1 | Intel(R) Core(TM) i7 CPU 860@2.80GHz | 4 | 2,800 | 500 | 4 | Front-End |
| 2 | Intel(R) Core(TM) i7 CPU 860@2.80GHz | 4 | 2,800 | 500 | 4 | Back-End |
| 3 | Intel(R) Core(TM) i7 CPU 860@2.80GHz | 4 | 2,800 | 500 | 4 | Back-End |
| 4 | Intel(R) Core(TM) i7 CPU 860@2.80GHz | 4 | 2,800 | 500 | 4 | Back-End |

We design a basic experiment of server performance and its throughputs as well. “Apache JMeter” [39], one of Apache projects, is a well-known web application measure performance tool. “Apache JMeter” is open source software, a 100% pure Java desktop application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions. Apache JMeter features include:

a. Can load and performance test many different server types: HTTP, HTTPS, SOAP, JDBC, LDAP, and JMS.
b. Mail - POP3(S) and IMAP(S)
c. Complete portability and 100% Java purity
d. Full multithreading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups.
e. Careful GUI design allows faster operation and more precise timings.
f. Caching and offline analysis/replaying of test results.
g. Highly Extensible

High-Performance Computing Cluster (HPCC) [40] is used to describe computing environments which utilize supercomputers and computer clusters to address complex computational requirements, support applications with significant processing time requirements, or require processing of significant amounts of data. It is also a benchmark for measure computing performance. The input parameters of HPCC can be considered with three key elements: **P** - the number of process rows, could be explained as CPU number, **Q** - the number of process columns could be explained as total server number, **N** - the order of the coefficient matrix A, it also called **Problem Size** in coming article. A formal formula can be described the required memory space of problem size:

$\boldsymbol{matrixSize}^2 \times 8\ bits = \boldsymbol{Required\ Memory}$ . The output of the HPCC is **Gflops** which means rate of execution for solving the linear system.

## 4.2. Networking Capability

In this section we evaluate the previous architecture by studying the effect of virtualizing the worker nodes and physical host. In order to quantify the different network throughput for local and remote nodes, Table 4-2 compares the transfer times, using the HTTP protocol, for different file sizes between the physical host and virtual machine. In same condition, Table 4-3 compares the throughputs via HTTP protocol under various file sizes and threads.

Table 4-2 A Comparisons of Physical Host and Virtual Machine Networking Performance

| ***Networking Transfer(KB/sec)*** | | | | |
|---|---|---|---|---|
| | | 20 Threads | 50 Threads | 100 Threads |
| Debain | 10MB | 70035.83 | 32750.53 | 36844.41 |
| Xen | | 68865.80 | 29646.50 | 35545.68 |
| Debain | 50MB | 48174.20 | 38210.20 | 25922.33 |
| Xen | | 46802.77 | 36307.15 | 24924.86 |

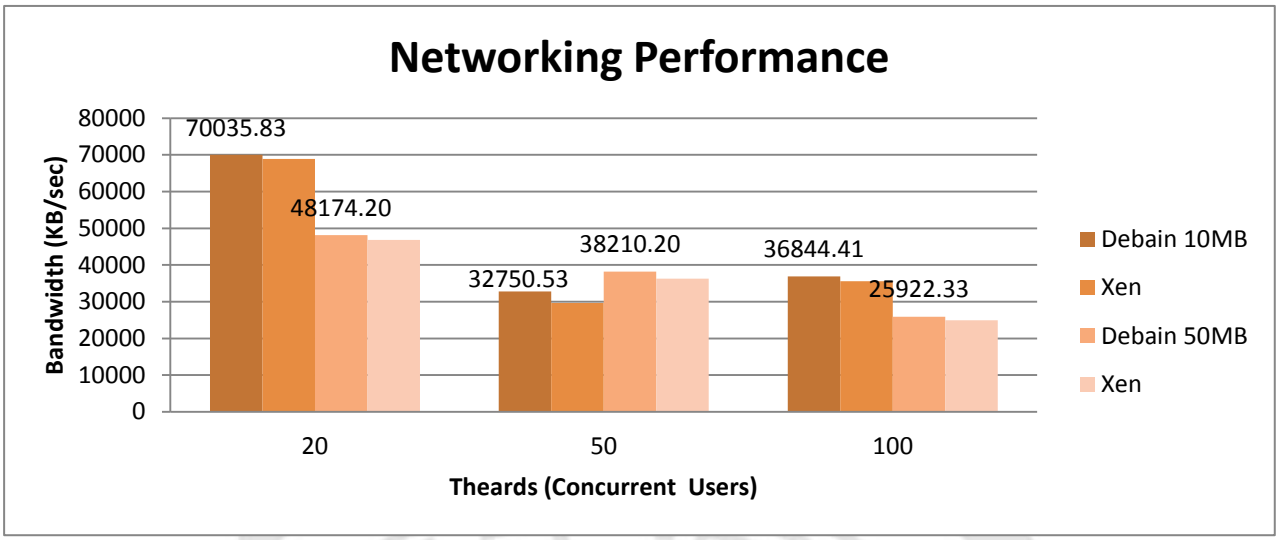


Figure 4-1. Physical Host and Virtual Machine Networking Performance

Table 4-3 Throughputs between Physical Host and Virtual Machine

| ***Throughputs*** | | | | |
|---|---|---|---|---|
| | | **20** Threads | **50** Threads | **100** Threads |
| Debain | 10MB | 6.80 | 3.20 | 3.60 |
| Xen | | 6.73 | 2.90 | 3.47 |
| Debain | 50MB | 0.94 | 0.75 | 0.49 |
| Xen | | 0.91 | 0.71 | 0.49 |

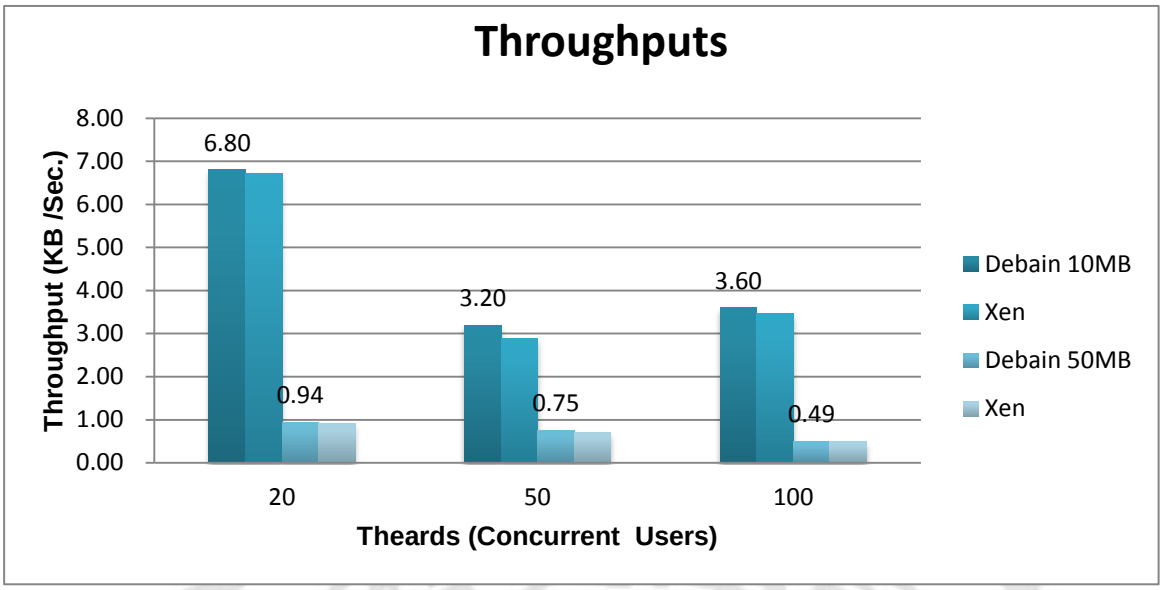


Figure 4-2. Throughputs between Physical Host and Virtual Machine

A significant result as shown previous Tables and Figures, the virtual machine performance is a little less than physical machine, it also match our expectancy.

## 4.3. Measurement Server Performance with HPCC Benchmark

Aim to the specific purpose, we need to know our server's performance as determine data to allocate how much resources in each site. Thus, we run hpcc test in this section to measure the server capabilities. In the coming Table, Experiments were conducted on eight nodes cluster, and each server owned two VMs on it. The nodes runs para-virtualization Linux 2.6.26-2 kernel as a privileged virtual machine on top of the Xen hypervisor. The guest virtual machines are configured to run the same version of the Linux kernel as that of the privileged one. They are constrained within 1 GB of main memory and allocated 1 CPU. Hence, total eight VMs were evenly distributed on four physical hosts, and we measure the floating points of the result.

We obtained result as shown Table 4-4, while the problem size was increased, computing

resource was also coming to physical limitations and, of course, it impacted the outcome data. You can see the curve is going gentle during problem size 19,000 – 25,000. Oppositely, it rapidly rose in the beginning of small problem sizes.

Table 4-4 Results of Running HPCC

| ***Results of Running HPCC on 8 VMs Input*** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **P.S(K)** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| G.FP | 3.68E+00 | 8.05E+00 | 1.21E+01 | 1.61E+01 | 1.98E+01 | 2.30E+01 | 2.61E+01 | 2.87E+01 | 3.13E+01 | 3.32E+01 |
| **P.S(K)** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** |
| G.FP | 3.53E+01 | 3.74E+01 | 3.916E+01 | 4.07E+01 | 4.245E+01 | 4.387E+01 | 4.554E+01 | 4.669E+01 | 4.764E+01 | 4.916E+01 |
| **P.S(K)** | **21** | **22** | **23** | **24** | **25** | | | | | |
| G.FP | 5.022E+01 | 5.114E+01 | 5.213E+01 | 5.309E+01 | 5.389E+01 | | | | | |

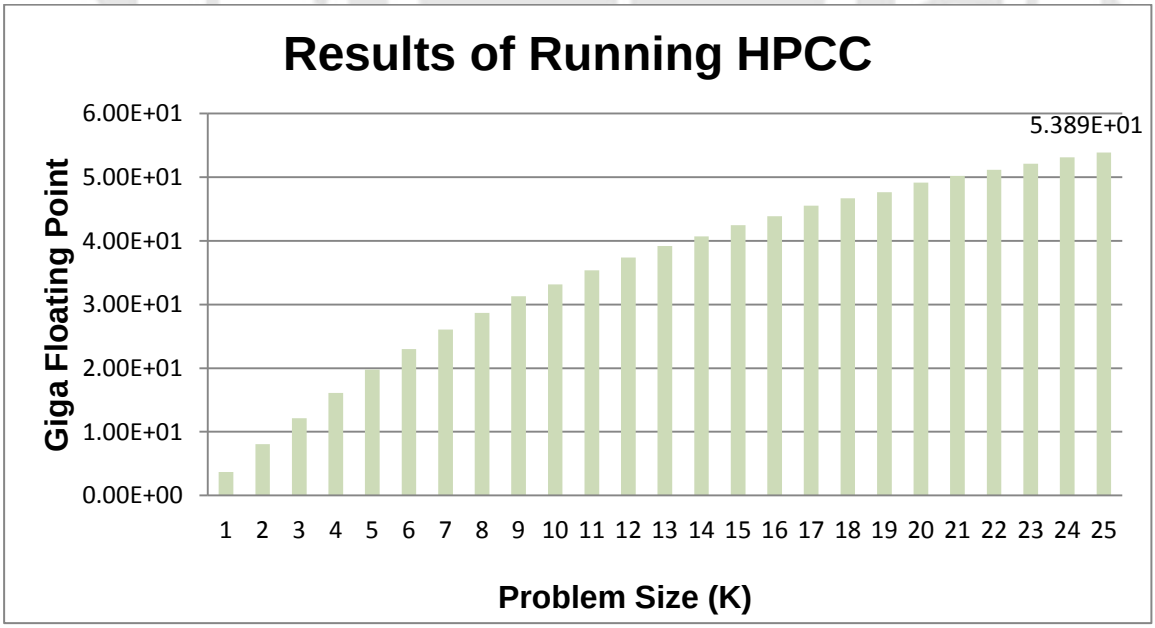


Figure 4-3. Results of Running HPCC

## 4.4. Virtual Machine Life Migration

We performed test migrations between an identical pair server machines, each with eight i7-Core 2.8GHz CPUs and 4GB memory. The machines are connected via switched Gigabit Ethernet. Before the migrate, demon required 1G space on each host, thus, the maximum available memory space was 3G for each host. There was only one virtual machine on Host-A, and zero VMs on Host-B, the VM on Host-A cost 1 G memory space, and we migrated the VM from Host-A to Host-B, you can see the memory usage variance in Figure 4-4 and Figure 4-5.

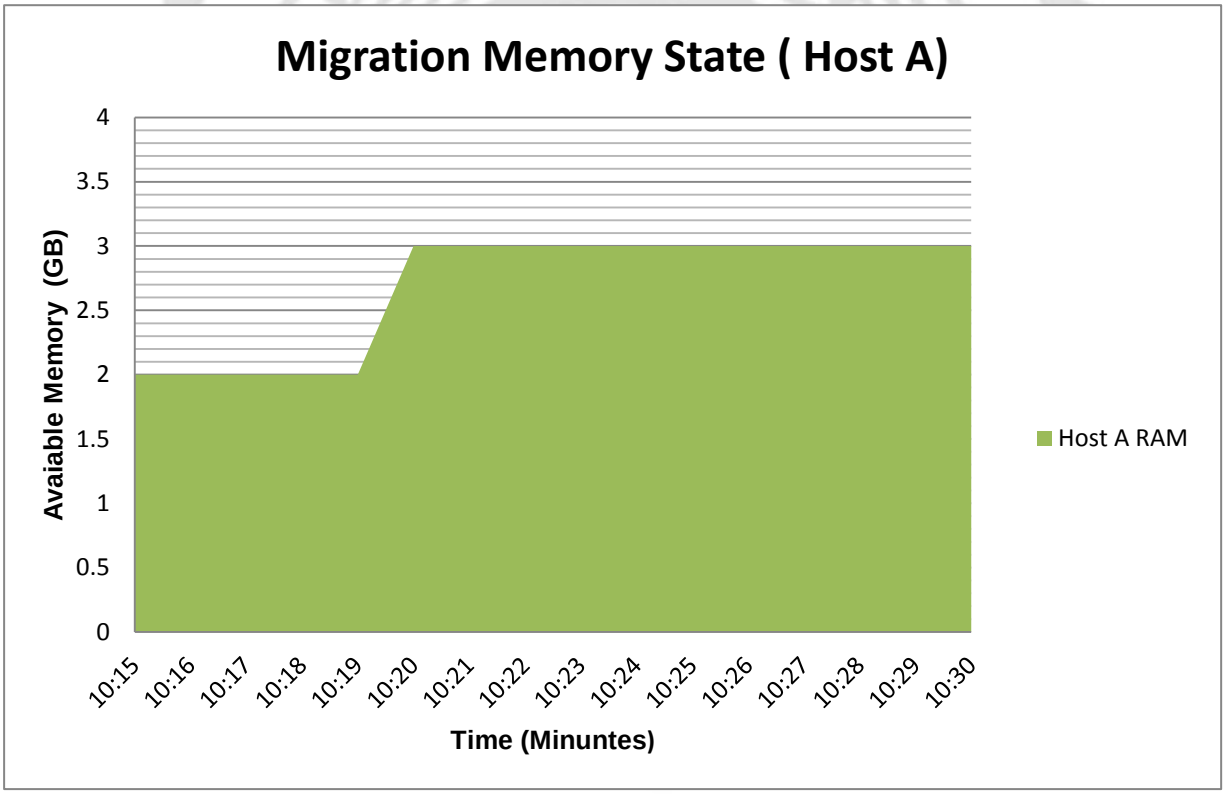


Figure 4-4. Host-A Migration Memory State

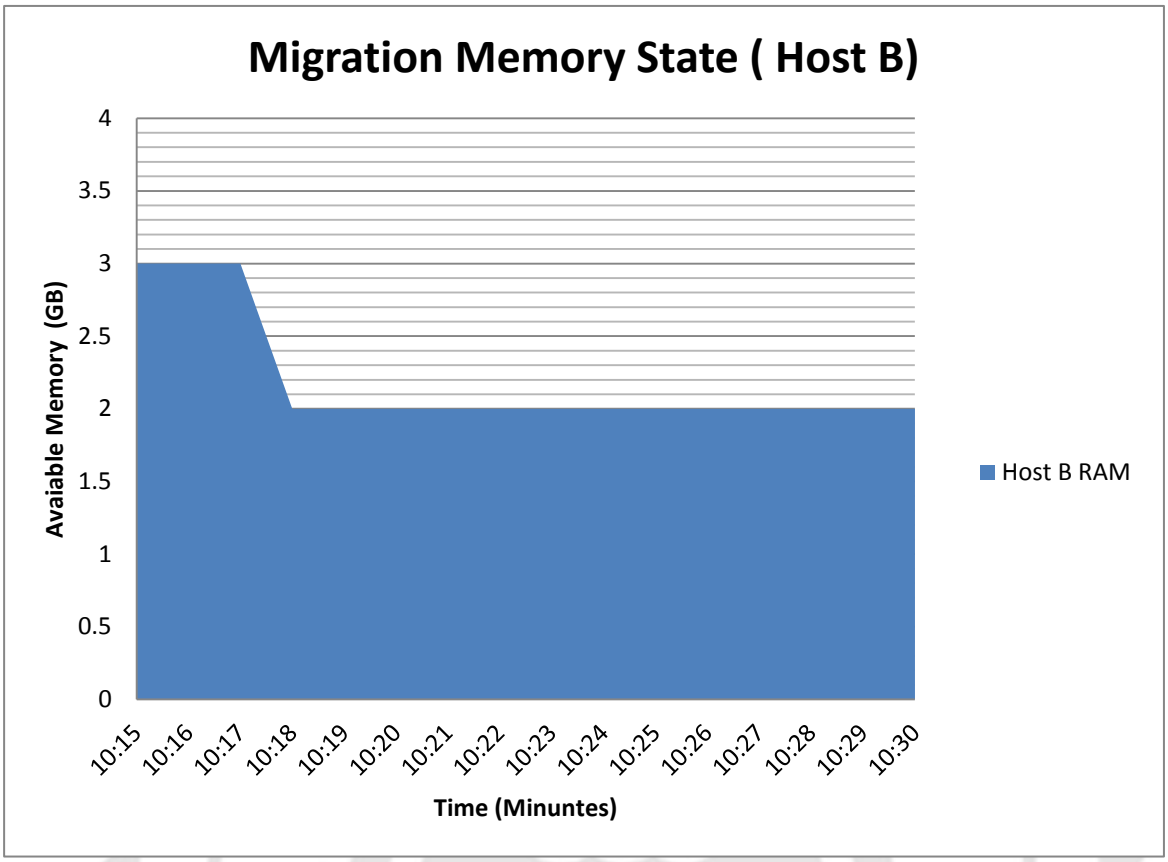


Figure 4-5. Host-B Migration Memory State

## 4.5. Hadoop Namenode Failover

In this experiment we gave below settings, Table 4-5 and Table 4-6, to build HDFS on virtual machine, it content one live node, and 28.61 GB spaces. In this scenario, we try to monitor the HDFS failover while downloading from it. Another tool for this test is FUSE [41], this tool allow user operate HDFS as local disk. Through the virtual IP setting we could get the HDFS information as shown Figure 4-6.

Table 4-5 Planned Hosts

| NO | Hostname | IP Address |
|---|---|---|
| 1 | debian-ha1 | 192.168.123.211 |
| 2 | debian-ha2 | 192.168.123.212 |
| Virtual | hadoop.namenode | 192.168.123.210 |

Table 4-6 Part of Properties of hdfs-site.xml

| Property | Value | Comments |
|---|---|---|
| dfs.data.dir | /drbd/hadoop/hdfs/data | On DRBD replicated volume |
| dfs.name.dir | /drbd/hadoop/hdfs/namenode | On DRBD replicated volume |
| fs.default.name | hdfs:// hadoop.namenode:8020 | Shared virtual name |
| mapred.job.tracker | hadoop.namenode:8021 | Shared virtual name |

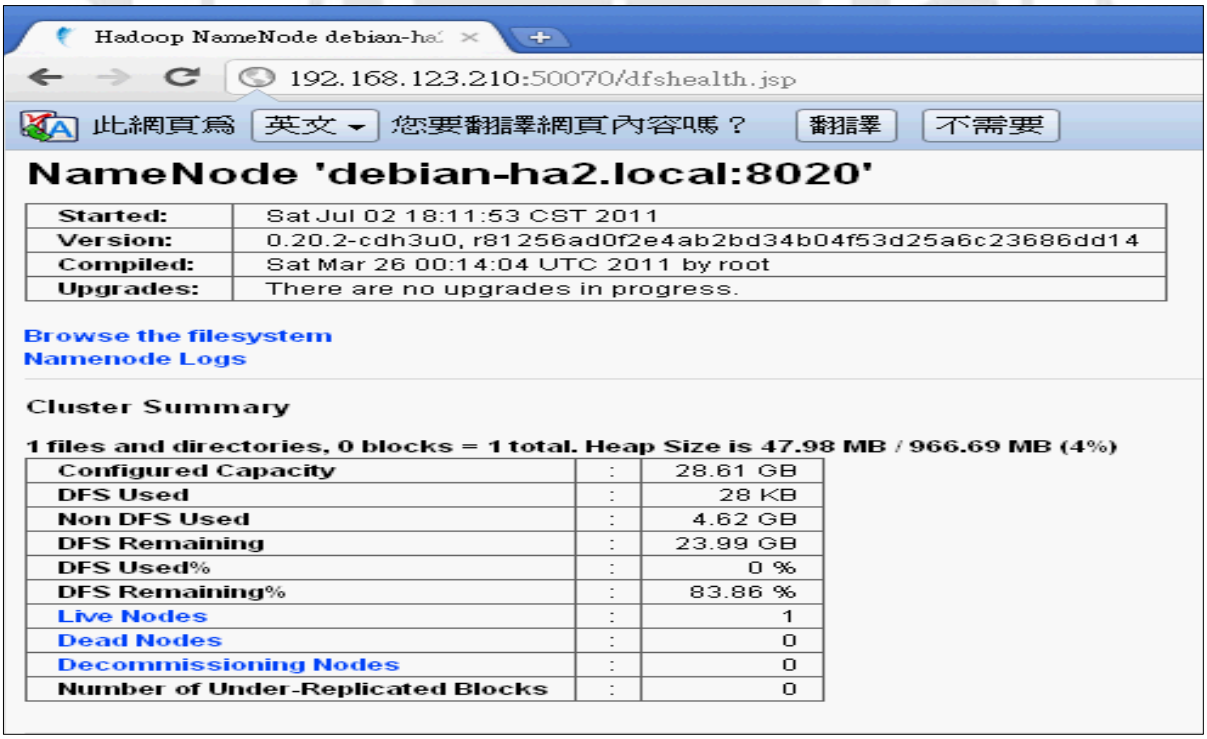


Figure 4-6. Lab Hadoop HDFS Information

While the downloading began from HDFS, we terminated the primary node (debian-ha1) and as our expectation, the downloading action was disconnected, after about 10-20 seconds, Namenode was resumed on debian-ha2 automatically. This result shows our design is only working on Active/Standby state. However, due to we still keep the metadata controlled by DRBD, the entire HDFS would not crash under unexpected system outages. It is really an enhancement for Hadoop Namenode issue, because lots of issues are talking about Namenode fail problems after unexpected system shutdown.

## 4.6. VFT Experimental

In this scenarios, we design an experiment to validate the virtual machine will migrate automatically or not if the host is off-line. **Service IP**: stand for provide service channel to external users, users through this IP to access services, it also name VIP in DRBD speaking. **Node2** is primary node, **Node1** is secondary. The different between primary and secondary is secondary will take over the service if primary node is done. **Debian1**, **Debian2** and **Debian3** are the physical hosts, Node2 is living on Debian 1, and the secondary node, Node1, is living on Debian2.

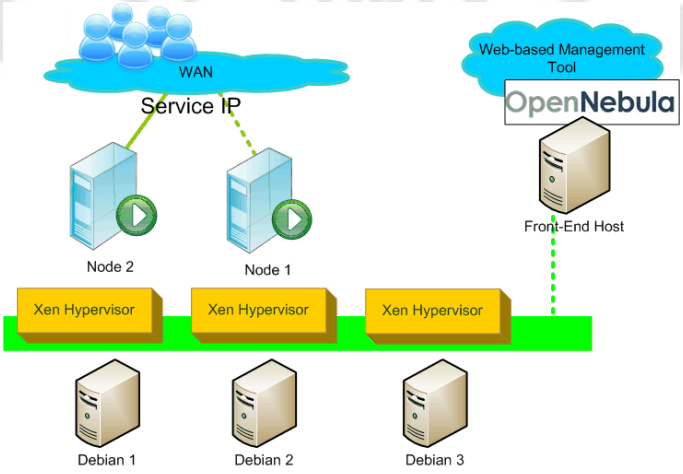


Figure 4-7. VFT Experiment Environment

In Figure 4-8, t shows that after Debian1 is disconnected that does not impassion the Service IP terminated, and it might only get lost one pack during the failover behavior enabled. The reason is that entire system is under our VFT mechanism that secondary is replaced primary node immediately. Finally, about 5-7 minutes the Node2 reboot automatically and become secondary.

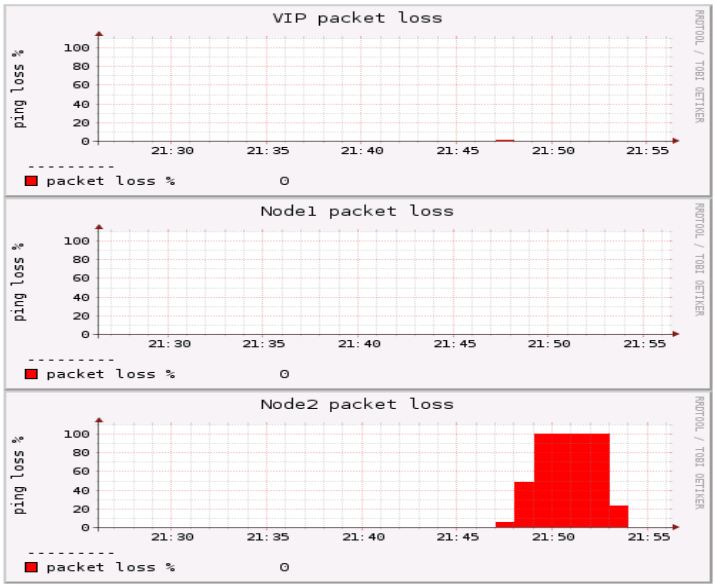


Figure 4-8. Ping Loss Measurement

The benefit of VFT mechanism is obtained a shortest downtime time. Although, we cannot guarantee the data without losing while downtime occurring. However, we still decrease the downtime, provided a low-cost solution for enterprise.

# Chapter 5

# Conclusions and Future Work

## 5.1. Concluding Remarks

High-Availability (HA), we proposed in this thesis, was to achieve Hadoop Namenode Active-Standby architecture. Under this architecture, the service can be failover since the primary node was failed. The most improvement was if you keep at last three physical hosts available, then the primary and secondary nodes would be always existed. Therefore, there are four main key features in this work, first is Xen Hypervisor, second is OpenNebula, third is DRBD with Heartbeat component, and the last is our VFT mechanism. Each of components is important and indispensable in our architecture. Systems continuous availability means comparatively and higher priced, and most has carefully implemented specialty designs that eliminate any single point of failure and allow online hardware, network, operating system, middleware, and application upgrades, patches, and replacements. In another word, the series high reliability also must be dependent on building a good human behavior institution. However, the future goal of this thesis is to extend our fault-tolerance work beyond failure management in order to enable better utilization of virtualization cluster resources.

## 5.2. Future Work

Some known issues in this thesis, like checkpoint problem and data transferring interruption

problem must enhance in next step. For instance, we could set more checkpoints to validate exists server or service. In the other hand, the transfer interruption issue is including complex difficult issues, but we could consider solving it with a tool named Zookper. Regarding green issues, how to raise the energy saving rate will be an essential work today. And we believe those vision will be implemented under the OpenNebula architecture. Therefore, we need to design more patterns aim to various situations.

# Bibliography


[1] V. Chaudhary, C. Minsuk, J. P. Walters, S. Guercio, and S. Gallo, "A Comparison of Virtualization Technologies for HPC," in *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on*, 2008, pp. 861-868.

[2] M.-V. Rafael, S. M. Ruben, and M. L. Ignacio, "Elastic management of cluster-based services in the cloud," in *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, ed. Barcelona, Spain: ACM, 2009, pp. 19-24.

[3] C. Engelmann, S. L. Scott, C. Leangsuksun, and X. He, "Symmetric Active/Active High Availability for High-Performance Computing System Services: Accomplishments and Limitations," in *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, 2008, pp. 813-818.

[4] D. Turner and C. Xuehua, "Protocol-dependent message-passing performance on Linux clusters," in *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*, 2002, pp. 187-194.

[5] *Xen*. Available: http://www.xen.org/

[6] *Hadoop*. Available: http://hadoop.apache.org

[7] C. Ning, W. Zhong-hai, L. Hong-zhi, and Z. Qi-xun, "Improving downloading performance in hadoop distributed file systm," *JOURNAL OF COMPUTER APPLICATIONS*, vol. 30, 2010.

[8] R. L. Grossman, Y. Gu, M. Sabala, and W. Zhang, "Compute and storage clouds using wide area high performance networks," *Future Generation Computer Systems*, vol. 25, pp. 179-183, 2009.

[9] J. Shafer, S. Rixner, and A. L. Cox, "The Hadoop distributed filesystem: Balancing portability and performance," in *Performance Analysis of Systems & Software*

*(ISPASS), 2010 IEEE International Symposium on*, White Plains, NY 2010, pp. 122-133.

[10] G. Mackey, S. Sehrish, and W. Jun, "Improving metadata management for small files in HDFS," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, 2009, pp. 1-4.

[11] *Cloudera*. Available: http://www.cloudera.com

[12] L. Xuhui, H. Jizhong, Z. Yunqin, H. Chengde, and H. Xubin, "Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, 2009, pp. 1-8.

[13] T. White. (October 2010). *Hadoop: The Definitive Guide (Secondary ed.)*.

[14] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *ACM Trans. Comput. Syst.*, vol. 26, pp. 1-26, 2008.

[15] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, pp. 29-43, 2003.

[16] C. Engelmann, S. L. Scott, C. Leangsuksun, and X. He, "Active/active replication for highly available HPC system services," in *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, 2006, p. 7 pp.

[17] L. Fei-fei, Y. Xiang-zhan, and W. Gang, "Design and Implementation of High Availability Distributed System Based on Multi-level Heartbeat Protocol," in *Control, Automation and Systems Engineering, 2009. CASE 2009. IITA International Conference on*, 2009, pp. 83-87.

[18] J. Walters and V. Chaudhary, "A fault-tolerant strategy for virtualized HPC clusters," *The Journal of Supercomputing*, vol. 50, pp. 209-239, 2009.

[19] E. Vargas, "High Availability Fundamentals," ed: Sun Microsystems, Inc., Nov. 2000.

[20] G. Vallee, C. Engelmann, A. Tikotekar, T. Naughton, K. Charoenpornwattana, C. Leangsuksun, and S. L. Scott, "A Framework for Proactive Fault Tolerance," in *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, 2008, pp. 659-664.

[21] C.-W. Ang and C.-K. Tham, "Analysis and optimization of service availability in a HA cluster with load-dependent machine availability," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, pp. 1307-1319, 2007.

[22] M. Dejan, L. M. Liorente, and R. S. Montero, "OpenNebula: A Cloud Management Tool," *Internet Computing, IEEE*, vol. 15, pp. 11-14, 2011.

[23] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," presented at the Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009.

[24] P. Sempolinski and D. Thain, "A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 417-426.

[25] C.-T. Yang, H.-Y. Cheng, W.-L. Chou, and C.-T. Kuo, " A Dynamic Resource Allocation Model for Virtual Machine Managemant on Cloud," in *Symposium on Cloud and Service Computing* 2011.

[26] F. Piedad and M. Hawkins, *High availability, design, techniques and processes*: Prentice-Hall, Inc., Jan. 2001.

[27] *DRBD Official Site*. Available: http://www.drbd.org

[28] *Heartbeat - Linux High Availability*. Available: http://linux-ha.org/wiki/Heartbeat

[29] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt,

and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, pp. 164-177, 2003.

[30] W. v. Hagen, *Professional Xen Virtualization*, 2008.

[31] C.-T. Yang, C.-H. Tseng, K.-Y. Chou, and S.-C. Tsaur, "Design and Implementation of a Virtualized Cluster Computing Environment on Xen," presented at the The second International Conference on High Performance Computing and Applications, HPCA, 2009.

[32] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for HPC with Xen virtualization," presented at the Proceedings of the 21st annual international conference on Supercomputing, Seattle, Washington, 2007.

[33] R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "An elasticity model for High Throughput Computing clusters," *Journal of Parallel and Distributed Computing*, vol. 71, pp. 750-757, 2011.

[34] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE Internet Computing*, vol. 13, 2009.

[35] *OpenVZ*. Available: http://wiki.openvz.org/Main_Page

[36] *OpenNebula*. Available: http://www.opennebula.org

[37] Z. Hai, T. Kun, and Z. Xuejie, "An Approach to Optimized Resource Scheduling Algorithm for Open-Source Cloud Systems," in *ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual*, 2010, pp. 124-129.

[38] *RDDtool*. Available: http://oss.oetiker.ch/rrdtool/

[39] *Apache JMeter*. Available: http://jakarta.apache.org

[40] *HPCC*. Available: http://icl.cs.utk.edu/hpcc

[41] *MountableHDFS*. Available: http://wiki.apache.org/hadoop/MountableHDFS

# Appendix

## A. Hadoop HA Setup and Configuration

HA Setup and Configuration, below list is our planning hosts

| NO | Hostname | IP Address |
|---|---|---|
| 1 | debian-ha1 | 192.168.123.211 |
| 2 | debian-ha2 | 192.168.123.212 |
| Virtual | hadoop.namenode | 192.168.123.210 |

Properties that will be defined as part of our hadoop-site.xml:

| Property | Value | Comments |
|---|---|---|
| dfs.data.dir | /drbd/hadoop/hdfs/data | On DRBD replicated volume |
| dfs.name.dir | /drbd/hadoop/hdfs/namenode | On DRBD replicated volume |
| fs.default.name | hdfs:// hadoop.namenode:8020 | Shared virtual name |
| mapred.job.tracker | hadoop.namenode:8021 | Shared virtual name |

Regarding to the HA setup comes down to six parts:

a. Install JDK 6 package

b. Configure networking

c. DRBD and Heartbeat installation

d. Configure DRBD

e. Install Hadoop

f. Configure Heartbeat

## A.1. Install JDK 6 package

Consider the only version of Java JDK that should be used with Hadoop is Sun's own. It has been a well collection in http://www.oracle.com/technetwork/java/index.html. At the time of this writing, the latest version is jdk-6u22-linux-i586.

```
# chmod +x jdk-6u22-linux-i586-rpm.bin

# ./jdk-6u22-linux-i586-rpm.bin
```

```
admin@debian-ha1:~$ java -version
java version "1.6.0_22"
Java(TM) SE Runtime Environment (build 1.6.0_22-b04)
Java HotSpot(TM) Client VM (build 17.1-b03, mixed mode, sharing)
```

## A.2. Configure networking

The following is an example from our systems.

Edit the file /etc/hosts

```
127.0.0.1       localhost

10.1.1.211      debian-ha1

10.1.1.212      debian-ha2

192.168.123.210 hadoop.namenode
```

Edit the file /etc/network/interface:

```
# This file describes the network interfaces available on your system

# and how to activate them. For more information, see interfaces(5).


# The loopback network interface

auto lo
```

```
iface lo inet loopback


# The primary network interface
allow-hotplug eth0
allow-hotplug eth1
iface eth0 inet static
        address 192.168.123.211
        netmask 255.255.255.0
        network 192.168.123.0
        broadcast 192.168.123.255
        gateway 192.168.123.254
        dns-nameservers 168.95.1.1
        dns-search csie.thu.edu.tw

iface eth1 inet static
        address 10.1.1.211
        netmask 255.255.255.0
```

Finally, reboot the system or restart networking:

## A.3. DRBD and Heartbeat installation

DRBD (including its kernel module) and Heartbeat are part of the “extras” repository:

```
# apt-get -y install drbd82 kmod-drbd82 heartbeat
```

Sometimes the installation of the Heartbeat package fails on the first try. Just try again; it may work for you the second time.

## A.4. Configure DRBD

Before continuing with the DRBD configuration, we highly recommend reading through the documentation and reviewing examples to get a clear understanding of the architecture and intended goals: http://www.drbd.org/docs/about/.

The following /etc/drbd.conf file is created on both nodes:

```
global {     usage-count yes;    }
common {     syncer { rate    30M; }    }
resource r0 {
   protocol C;
   startup {
      wfc-timeout 0;
      degr-wfc-timeout 120;
   }
   disk {
      on-io-error     detach;
      # no-disk-flushes;
      # no-md-flushes
      # size 1G;
    }
   net {
    }
on debian-ha1 {
    device       /dev/drbd0;
    disk         /dev/sdb1;
```

```
    address      10.1.1.211:7789;
    meta-disk internal;
  }
on debian-ha2 {
    device       /dev/drbd0;
    disk         /dev/sdb1;
    address      10.1.1.212:7789;
    meta-disk internal;
  }
}
admin@debian-ha1:/etc/network$ clera
-bash: clera: command not found
admin@debian-ha1:/etc/network$ clear
admin@debian-ha1:/etc/network$ cat /etc/drbd.conf
global {
  usage-count yes;
}
common {
  syncer { rate   30M; }
}
resource r0 {
  protocol C;
  startup {
    wfc-timeout 0;
    degr-wfc-timeout 120;
```

```
    }
    disk {
        on-io-error     detach;
        # no-disk-flushes;
        # no-md-flushes
        # size 1G;
      }
    net {
      }
on debian-ha1 {
      device      /dev/drbd0;
      disk        /dev/sdb1;
      address     10.1.1.211:7789;
      meta-disk internal;
    }
on debian-ha2 {
      device      /dev/drbd0;
      disk        /dev/sdb1;
      address     10.1.1.212:7789;
      meta-disk internal;
    }
}
```

## A.5. Hadoop Installation

Hadoop is OpenSource software; there is lot of resource on internet. We used the web-based configurator provided by Cloudera (https://my.cloudera.com/), which builds an RPM containing repos for your custom configuration and the rest of their distribution. The resulting RPM is then installed on BOTH master nodes.

Add APT List

```
#vi /etc/apt/sources.list
        #willie 06.05 Cloudra Hadoop
        deb http://archive.cloudera.com/debian maverick-cdh3 contrib
        deb-src http://archive.cloudera.com/debian maverick-cdh3 contrib
```

APT update

```
#apt-get update
```

CURL Install

```
#apt-get install curl
```

GPG Key Install

```
#curl -s http://archive.cloudera.com/debian/archive.key | sudo apt-key add
```

Hadoop Install

```
#apt-get install hadoop hadoop-conf-pseudo hadoop-jobtrackerhadoop-namenode hadoop-0.20-native hadoop-secondarynamenode
```

Cancel Auto Run

```
#update-rc.d -f hadoop-namenode remove
#update-rc.d -f hadoop-secondarynamenode remove
#update-rc.d -f hadoop-jobtracker remove
```

## A.6. Heartbeat Configuration

There are many options available for the Heartbeat configuration. Here, we attempt to show only the basics that how to enable the Hadoop via Heartbeat. There are three key files that we edit to configure the Heartbeat package:

/etc/ha.d/ha.cf

/etc/ha.d/haresources

/etc/ha.d/authkeys

Create Soft Link

```
#cd /etc/ha.d/resource.d
    #ln -s /etc/init.d/hadoop-0.20-namenode hadoop-namenode
    #ln -s /etc/init.d/hadoop-0.20-jobtracker hadoop-jobtracker
```

/etc/ha.d/ha.cf

```
## start of ha.cf
logfile /var/log/ha-log
logfacility     local0

keepalive 2          #Detection period
warntime 5
deadtime 20
initdead 120
#hopfudge 1

udpport 694          #Using UDP 694
auto_failback off    #if failback, resume to master
```

```
#baud 19200
bcast eth1                #using eth1, to be the heartbeat network card
ucast eth0 192.168.123.211
ucast eth1 10.1.1.211

node debian-ha1      #Node 1, Server Name
node debian-ha2      #Node 2, Server Name

ping 192.168.123.254          #Ping our Gateway, check heart self

respawn hacluster /usr/lib/heartbeat/ipfail
apiauth ipfail gid=haclient uid=hacluster
## end of ha.cf
```

/etc/ha.d/haresources

```
#vim /etc/ha.d/haresources
debian-ha1  192.168.123.210/24  drbddisk::r0  Filesystem::/dev/drbd0::/drbd::ext3::noatime
hadoop-namenode
```

Heartbeat Restart

```
#/etc/init.d/heartbeat stop
#/etc/init.d/heartbeat start
```