Author Query Form

Journal: SUPE **Article ID:** SUPE1045	**Please send your responses together with your list of corrections via web (preferred), or send the completed form and your marked proof to:** Mokslininku 2a, LT-08412 Vilnius, Lithuania fax: +370 5 2784 091 e-mail: vtexspr-corrections@vtex.lt

Dear Author,

During the preparation of your manuscript for typesetting, some questions have arisen. These are listed below.

Queries and/or remarks

Location in article (page/line)	Query / remark	Response
19/856, 865; 20/894, 902	If possible, please update Refs. [7,13,30,34]. DOI numbers would be sufficient.	
20/909	Please cite or delete the uncited references. You will find a special note in the margin by each uncited reference item.	

Many thanks for your assistance

Metadata of the article that will be visualized in Online First

Please note: Images will appear in color online but will be printed in black and white.

Journal Name	The Journal of Supercomputing	
Article Title	On improvement of cloud virtual machine availability with virtualization fault tolerance mechanism	
Copyright holder	Springer Science+Business Media New York This will be the copyright line in the final PDF.	
Corresponding Author	Family name	Yang
	Particle	
	Given Name	Chao-Tung
	Suffix	
	Division	Department of Computer Science
	Organization	Tunghai University
	Address	Taichung, 40704, Taiwan
	E-mail	ctyang@thu.edu.tw
Author	Family name	Liu
	Particle	
	Given Name	Jung-Chun
	Suffix	
	Division	Department of Computer Science
	Organization	Tunghai University
	Address	Taichung, 40704, Taiwan
	E-mail	jcliu@thu.edu.tw
Author	Family name	Hsu
	Particle	
	Given Name	Ching-Hsien
	Suffix	
	Division	Department of Computer Science and Information Engineering
	Organization	Chung Hua University
	Address	Hsinchu, 30010, Taiwan
	E-mail	chh@chu.edu.tw
Author	Family name	Chou
	Particle	
	Given Name	Wei-Li

	Suffix	
	Division	Department of Computer Science
	Organization	Tunghai University
	Address	Taichung, 40704, Taiwan
	E-mail	nagage@gmail.com

Schedule	Received	
	Revised	
	Accepted	

Abstract	Virtualization, particularly in the field of cloud computing, is a common strategy to improve existing computing resources. Hadoop, one of the Apache projects, is designed to scale up from single servers to thousands of machines, each offering local computation and storage capabilities. However, how to guarantee both stability and reliability of virtualization have become important topics. In this article, to reach this goal we used current open-source software and platforms, for instance, the Xen-Hypervisor virtualization technology, and the OpenNebula virtual machines management tool. After extending components capabilities, we developed a mechanism to support our ideas and reached high availability with Hadoop that is also called as virtualization fault tolerance (VFT). We considered a practical problem, i.e., the single-point-of-failure issue that occurs frequently in virtualization systems, and the experimental results confirm that the downtime interval can be greatly shortened even if failure occurred. As a result, VFT is useful not only for Hadoop applications, but also for more areas in cluster-based systems.

Keywords	High availability − Cloud computing − IaaS − Virtualization − Virtualization fault tolerance
Footnotes	

« SUPE 11227 layout: Small Extended v.1.1 file: supe1045.tex (Nadia) class: spr-small-v1.3 v.2013/11/19 Prn:2013/11/20; 16:09 p. 1/20»
« doctopic: OriginalPaper numbering style: ContentOnly reference style: basic»

AUTHOR'S PROOF

J Supercomput
DOI 10.1007/s11227-013-1045-1

On improvement of cloud virtual machine availability with virtualization fault tolerance mechanism

Chao-Tung Yang · Jung-Chun Liu · Ching-Hsien Hsu · Wei-Li Chou

© Springer Science+Business Media New York

UNCORRECTED PROOF

Abstract Virtualization, particularly in the field of cloud computing, is a common strategy to improve existing computing resources. Hadoop, one of the Apache projects, is designed to scale up from single servers to thousands of machines, each offering local computation and storage capabilities. However, how to guarantee both stability and reliability of virtualization have become important topics. In this article, to reach this goal we used current open-source software and platforms, for instance, the Xen-Hypervisor virtualization technology, and the OpenNebula virtual machines management tool. After extending components capabilities, we developed a mechanism to support our ideas and reached high availability with Hadoop that is also called as virtualization fault tolerance (VFT). We considered a practical problem, i.e., the single-point-of-failure issue that occurs frequently in virtualization systems, and the experimental results confirm that the downtime interval can be greatly shortened even if failure occurred. As a result, VFT is useful not only for Hadoop applications, but also for more areas in cluster-based systems.

Keywords High availability · Cloud computing · IaaS · Virtualization · Virtualization fault tolerance

C.-T. Yang (✉) · J.-C. Liu · W.-L. Chou
Department of Computer Science, Tunghai University, Taichung, 40704 Taiwan
e-mail: ctyang@thu.edu.tw

J.-C. Liu
e-mail: jcliu@thu.edu.tw

W.-L. Chou
e-mail: nagage@gmail.com

C.-H. Hsu
Department of Computer Science and Information Engineering, Chung Hua University, Hsinchu, 30010 Taiwan
e-mail: chh@chu.edu.tw

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1
PDF-OUTPUT

AUTHOR'S PROOF

1 Introduction

Virtual machines (VMs) have been popular in the recent two decades; the annual growth rate of VM applications has significantly increased [1–5]. In addition to many VM related products offered by various vendors, emerging VM applications are found in different fields, such as green energy saving, cluster management, and behavior detection. The virtualization technology provides not just secondary, but key applications in many fields. Along with the expending of the virtualization technology, the VM guest operating system (Guest OS) continues to improve efficiency in operation.

Hadoop [6–13] was inspired by Google's MapReduce and Google File System (GFS) [14, 15]. The Hadoop cluster includes multiple worker nodes and a single master that consists of a JobTracker, TaskTracker, NameNode, and DataNode. The Hadoop Distributed File System (HDFS) uses information of rack names when replicating data and tries to keep different copies of data on different racks. The goal is to reduce the impact of a rack's power outage or switch off failure; thus, even when these events occur, data may still be readable. However, it takes a long time to restart the system from failure.

For most people, it is a big challenge to embrace a new technology; the learning curve is daunting, and issues of reliability and stability are even worse. Hadoop, like the other distribution systems, allows users to operate complex computing with back-end resources and be in charge of metadata links or resource allocation work in the front-end. Developers could use these features to achieve service aims. In this paper, we conducted Hadoop NameNode running on virtual machines and developed a high availability mechanism for NameNode. The HDFS instance requires one unique server, i.e., the name node; thus, there is a single point of failure for an HDFS installation. If the name node goes down, the file system will be offline. When it comes back, the name node must replay all outstanding operations, which could overtake a big cluster for half an hour. The file system includes a secondary NameNode, which regularly connects with the primary NameNode and takes snapshots of the primary NameNode's directory information, which is later saved to local/remote directories. These checkpoint images can be used to restart a failed primary NameNode without replaying the entire repertoire of the file system action. The edit log creates an up-to-date directory structure as well.

Various challenges are faced while developing a distributed application [3, 16–21]. The first problem is hardware failure. If more pieces of hardware are used, the chance to fail becomes even higher. The second problem is that most analysis tasks need to combine data in some way, i.e., data read from one disk may need to be combined with data read from other disks. HDFS replicates redundant copies of data kept by the system, so that in the event of failure, another copy of data is available. This is mostly like the way RAID works. MapReduce offers a programming model that abstracts problems from disk reads and writes as computations over sets of keys and values.

However, currently Hadoop does not support automatic recovery for NameNode failure, a well-known and recognized single point of failure in Hadoop. As mentioned in the Hadoop official site [6] that if the NameNode machine fails, manual

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

AUTHOR'S PROOF

intervention is necessary. Currently, automatic restart and failover of the NameNode software to another machine is not supported. Hadoop infrastructure has become a critical part of day-to-day business operations. As such, it is 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. Moreover, it is easy for us to follow the best practice of offloading the secondary NameNode data to an NFS mount to protect metadata, ensuring that processes are constantly available for job execution and data retrieval. We have leveraged some existing well tested components that are available and commonly used in Linux systems today. Our solution, called as Virtualization Fault Tolerance (VFT), primarily makes use of Distributed Replicated Block Device (DRBD) [1] from LINBIT, and Heartbeat from the Linux-High Availability (HA) project. The combination of these projects provides us with a reliable and highly available solution to address current limitations.

Virtualization is used as a solution not only to improve service flexibility, but also to consolidate workloads and enhance utilization of the server. A virtualized system can be dynamically adapted to clients' demands by deploying new virtual nodes when demands increase, and powering off and consolidating virtual nodes during periods of low demand. In this paper, we employed the virtual machine management tool, OpenNebula [22–24], to manage virtual machines, and combined it with other open source resources to achieve the goal of high availability for Hadoop NameNode.

This paper is organized as follows. First, we start with background reviews and related works in Sect. 2. Section 3 describes the system implementation, shows how to design the VFT mechanism, and presents the interface of our virtual machine management tool. In Sect. 4, we design some scenarios to prove our system and mechanism. Finally, Sect. 5 outlines main conclusions and the future work.

2 Background review and related work

2.1 Apache project: HADOOP

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

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 on cloud configurations that remove as many single points of failure as possible and that are inherently designed with a specific effort on operational continuity, redundancy, and fail-over capability.

Floyd Piedad et al. [26] presented availability levels and measurement in the HA field. They indicated that IT must understand the levels of availability required by

UNCORRECTED PROOF

AUTHOR'S PROOF

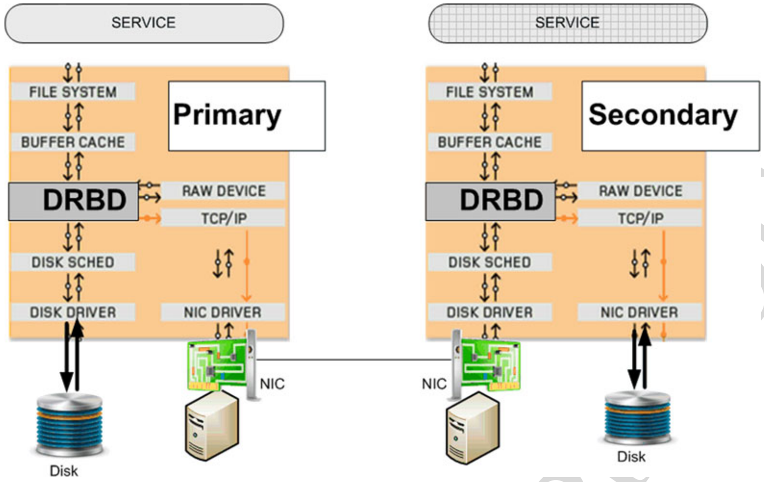

Fig. 1 DRBD architecture

users, and users must understand the costs to these targets. Of all availability levels, continuous availability is the most challenging and expensive to provide; in our work, we take this topic forward and try to make HA feasible.

2.3 Fault tolerance technology

In this paper, we consider DRBD with Heartbeat to be a good fault tolerance solution technology. 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 is designed as a device building block to form a HA cluster. This is done by mirroring a whole block device via a specified network. The DRBD technology can be understood as a network RAID-1. Figure 1 displays the entire DRBD architecture. The service, including its IP address, can be migrated to other nodes at any time, either due to a failure of the active node or as an administrative action. In HA speaking, 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].

DRBD's core functionality is implemented by 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 DRBD is extremely flexible and versatile, a replication solution is suitable for adding high availability to any other applications. Heartbeat [28] is daemon software that provides cluster infrastructure (communication and membership) services to its clients. It allows clients to be aware of presence of peer processes on other machines and to effortlessly exchange messages with them [30]. As shown in Fig. 2, DRBD with Heartbeat, which plays a very important role in our system, is a fault tolerance solution in Linux based OS.

UNCORRECTED PROOF

« SUPE 11227 layout: Small Extended v.1.1 file: supe1045.tex (Nadia) class: spr-small-v1.3 v.2013/11/19 Prn:2013/11/20; 16:09 p. 5/20»
« doctopic: OriginalPaper numbering style: ContentOnly reference style: basic»

AUTHOR'S PROOF

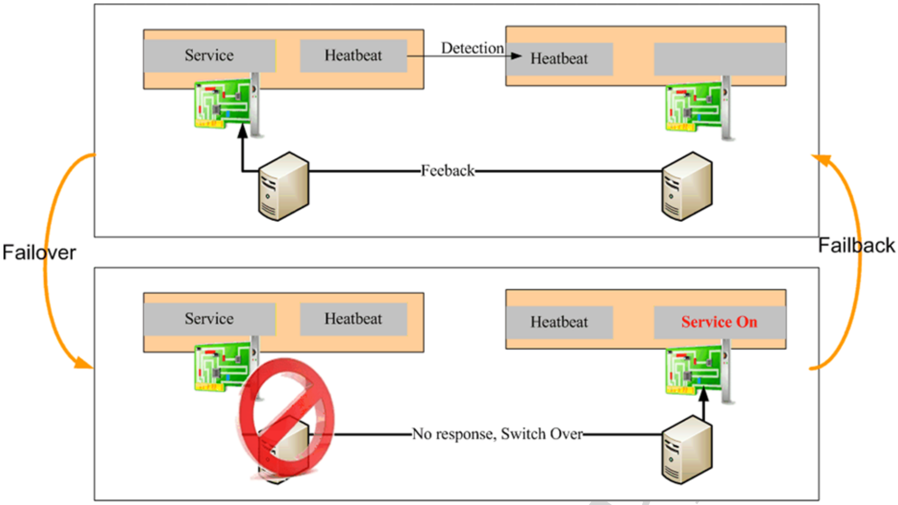

Fig. 2 DRBD with heartbeat

2.4 Virtualization technologies

Virtualization technology [1, 5, 18, 29–32] is an interesting solution to implement cluster-based servers to overcome cluster related problems. Cluster nodes can be virtualized through some virtualization platforms (Xen, KVM, VMWare, etc.) and managed by an efficient virtual machine manager. A provisioning model is incorporated for dynamically deploying new virtual cluster nodes when the user demand increases, and consolidating virtual nodes when it decreases. Virtualization runs multiple virtual machines on a single physical machine, with each virtual machine sharing the resources of that physical computer across multiple environments.

Virtualization is simply the logical separation to request services from the physical resources that actually provide them. In practical terms, virtualization offers the ability to run applications, operating systems, or system services in a logically distinct system environment that is independent of any specific physical computer system. Obviously, all of these have to run on a certain computer system at any given time, but virtualization provides 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. Through virtualization, one can execute applications under many operating systems, manage IT more efficiently, and share a lot of computing resources with other computers.

2.5 Virtual machine management

A key component in the scenario of virtualization is the virtual machine management system. The VM manager provides a centralized platform for efficient and automatic deployment, control, and monitoring of VMs in a distributed pool of physical resources. Usually, the VM manager also offers high availability capabilities

UNCORRECTED PROOF

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

AUTHOR'S PROOF

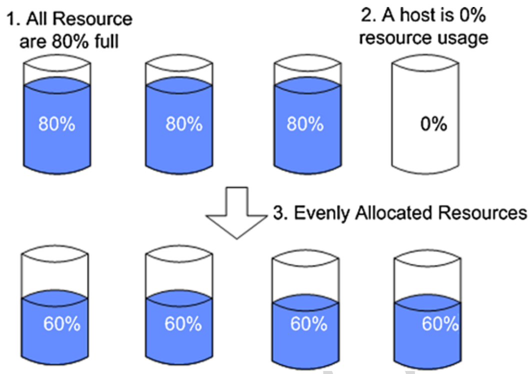

Fig. 3 Dynamic resource allocation

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 provision of virtual machines for cloud providing infrastructure-as-a-service. These various open-source projects provide important alternatives for those who do not wish to use a commercially provided cloud. In this paper, we employed OpenNebula to implement the research platform.

OpenNebula is a virtual infrastructure engine that enables the dynamic deployment and reallocation 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 from the physical infrastructure as well as the physical location. OpenNebula contains one front-end and multiple back-ends. 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 up. Communications between front-end and back-end employ Secure Shell (SSH). OpenNebula gives users a single access point to deploy virtual machines on a locally distributed infrastructure.

2.6 Dynamic resource allocation

In our previous paper, we proposed a Dynamic Resource Allocation algorithm (DRA) [25], which is one of the key components in this paper. In this work, we focus on enhancing Hadoop HA architecture problem; therefore, DRA is not described in detail here. However, the purpose of DRA is to achieve the best balance of resource allocation among physical machines. As shown in Fig. 3, to achieve the maximum efficiency the resource must be evenly distributed. In order to avoid computing resources centralizing on some specific physical machines, how to balance the resources becomes the most important issue.

2.7 Related works

Another choice to achieve fault tolerance is to use OpenVZ [35], which is container-based virtualization for Linux. OpenVZ creates multiple secure and isolated containers on a single physical server, enabling better server utilization and preventing applications from conflicting. J. Walters et al. [18], proposed to use both check-pointing

UNCORRECTED PROOF

AUTHOR'S PROOF

and replication in order to ensure the lowest possible check-pointing overhead. However, there are still some open issues about how to integrate check-pointing and fault-tolerance systems into common cluster batch schedulers. But they still provide us a nice practice to handle fault tolerance for virtualization on a single site.

G. Vallee et al. [20] proposed a framework to solve the fault tolerance issue. Such a framework enables the implementation of various fault tolerance policies, including policies presented in the literature that are not validated by experimentation; therefore, they presented a framework coupled with their fault tolerance simulator, and a complete solution for the study of proactive fault tolerance policies. Their framework prototype provides a single policy based on the Xen VM migration, but new policies are still under development. This is the reason why framework needs to be managed via VM management tools, such as OpenNebula [36]. As shown in this study, the Xen VM migration issue has been solved under our framework.

Regarding to the Fault Tolerance mechanism on Hadoop, a good solution was presented by Cloudera [11]. Cloudera focused on providing various Hadoop solutions. In 2009, Christophe Bisciglia presented an article, "Hadoop HA Configuration," which implemented Headbeat and DRBD to enhance Hadoop HA, and showed how to extend it for visualization.

H. Zhong et al. [37] proposed an optimized scheduling algorithm to achieve the optimization or suboptimization for cloud scheduling problems. In the same research, the authors investigated the possibility to allocate VMs in a flexible way to allow the maximum usage of physical resources. They used an Improved Genetic Algorithm (IGA) for the automated scheduling policy. IGA uses the shortest genes and introduces the idea of dividend policy in economics to select an optimal or suboptimal allocation for the VMs requests. This paper has inspired us to find an optimized algorithm to reach our goal.

Q. Chen et al. [38] proposed a Self-Adaptive MapReduce scheduling algorithm (SAMR) that dynamically computes progress of tasks and automatically adapts to continuously changing environments. SAMR tunes time weight of each stage of map and reduces tasks based on historical information to trace the progress of tasks and identify tasks that are in need of backup tasks.

3 System implementation

In this section, we introduce the system architecture and its components. OpenNebula plays a key role in the entire system; its most advantage is the Live Migration function that is lacked in other virtualization management tools. In addition to the Live Migration function from OpenNebula, we combined DRBD with Heart Beat to enhance high availability of the system.

3.1 System overview

The system was mainly constructed based on the official OpenNebula manual. The OpenNebula core orchestrates three different management areas: image and storage technologies (i.e., virtual appliance tools or distributed file systems) to prepare disk

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

AUTHOR'S PROOF

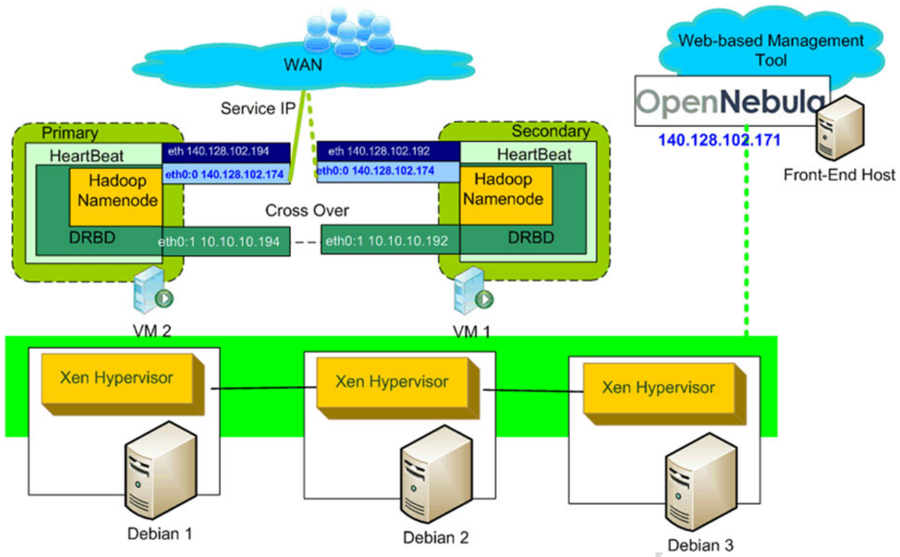

Fig. 4 System overview

images for VMs, the network fabric (such as Dynamic Host Configuration Protocol servers, firewalls, or switches) to provide VMs with a virtual network environment, and the underlying hypervisors to create and control VMs. The core performs specific storage, networking, or virtualization operations through pluggable drivers. Thus, OpenNebula is not tied to any specific environment but provides a uniform management layer regardless of the underlying infrastructure.

Figure 4 depicts an overview of our system architecture. As described, we built a cluster system with OpenNebula and provided users a web interface to manage virtual and physical machines. Our cluster system consists of four computers with same specifications; hardware of these computers is equipped with Intel i7 CPU 2.8 GHz, four gigabytes memory, 500 gigabytes disk, Debian operating system, and a gigabit switch to connect to the network.

As depicted in the figure, from the bottom to the top of the infrastructure: hosts are physical machines Debian $1 \sim 3$; Xen Hypervisors are suitable for Linux series OS; the following up are two VMs: VM 2 is the primary node, and VM 1 is the secondary node. But if we assume Hadoop NameNode is built on VM 1 as the primary node, VM 2 becomes the slave node of VM 1. Under the Heartbeat + DRBD mechanism, we used 5 IPs to deploy the system; two for Cross Over, two for identifying the primary and secondary, and one for service. Finally, on the top layer, as the key role of the entire design, OpenNebula provides a centralized platform as an efficient and automatic deployment to control and monitor VMs on a distributed pool of physical hosts. We also composed a web interface management tool via DRA and OpenNebula's components to manage VMs.

Due to limitation of the physical IP address, we built a private network environment in our laboratory. To enable the HA mechanism, some preliminary works needed to be done. First, we set the IPs on both virtual machines. IP 192.168.123.210

UNCORRECTED

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

AUTHOR'S PROOF

Table 1 VM2—primary node network setting

IP setting	Description
eth0 192.168.123.212	For identification of this machine
eth0:0 192.168.123.210	Service IP, controlled By Heartbeat to provide services for outside users
eth1 10.1.1.211	For data transfer controlled by DRBD

Table 2 VM1—secondary node network setting

IP setting	Description
eth0 192.168.123.212	For identification of this machine
eth0:0 192.168.123.210	Service IP, controlled by Heartbeat, and disabled when this machine is the secondary node
eth1 10.1.1.212	For data transfer controlled by DRBD

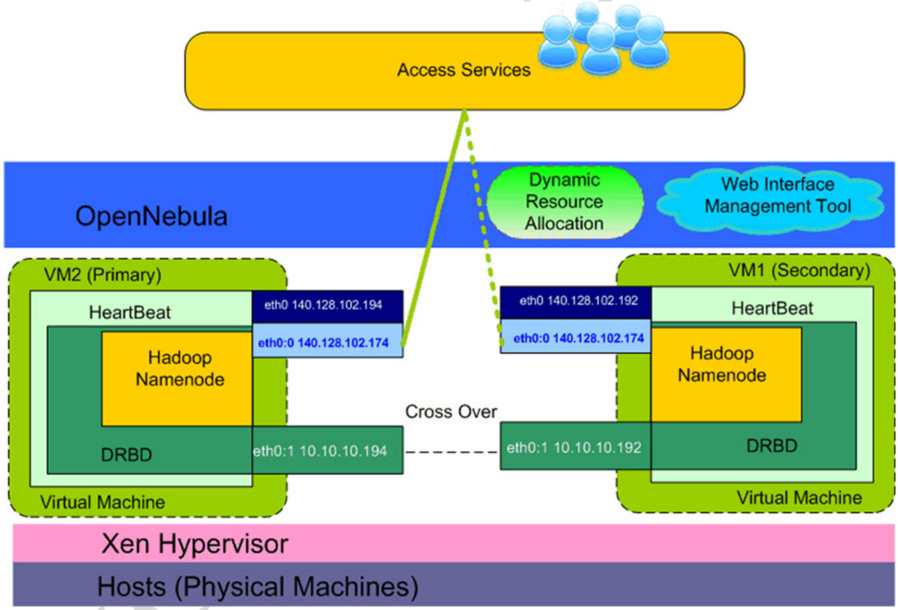

Fig. 5 Networking configuration of primary and secondary nodes

is the Service IP controlled by Heartbeat, and is used to provide services for users. In the configuration, VM 2 is the primary node (refer to Table 1 for its setting) and VM 1 is the secondary (refer to Table 2 for its setting), as shown in Fig. 5.

After downloading the DRBD package [30] and completely installing it, then we could start to set DRBD config file in both two nodes with the setting as shown in

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

« SUPE 11227 layout: Small Extended v.1.1 file: supe1045.tex (Nadia) class: spr-small-v1.3 v.2013/11/19 Prn:2013/11/20; 16:09 p. 10/20»
« doctopic: OriginalPaper numbering style: ContentOnly reference style: basic»

AUTHOR'S PROOF

Fig. 6 Part of drbd.conf content

```
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;
  }
}
```

Fig. 6. We show part of drbd.conf content in /etc/drbd.conf file. For the reminder, consistent setting is needed in both the primary and secondary nodes.

Use below commands to check the DRBD state: `#cat /proc/drbd` or `#drbdadm state r0`

There are many options available for the Heartbeat configuration. In the following, we will show our methods. There are three main files that we edited to configure the Heartbeat package:

- `/etc/ha.d/authkeys`
- `/etc/ha.d/ha.cf /`
- `/etc/ha.d/haresources`

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

`#chmod 0600 /etc/ha.d/authkeys`

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

UNCORRECTED PROOF

« SUPE 11227 layout: Small Extended v.1.1 file: supe1045.tex (Nadia) class: spr-small-v1.3 v.2013/11/19 Prn:2013/11/20; 16:09 p. 11/20»
« doctopic: OriginalPaper numbering style: ContentOnly reference style: basic»

AUTHOR'S PROOF

Fig. 7 Part of ha.cf content

```
## 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
```

Fig. 8 Part of drbd.conf content

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

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

Second, the ha.cf file is used to define the general settings of the cluster. Our example is shown in Fig. 7.

Finally, as shown in Fig. 8, the last file, ha resource, defines all cluster resources that will fail over from one node to the next. The resources include the Service IP addresses 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.

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

AUTHOR'S PROOF

3.2 Virtualization fault tolerant methodology

Our approach to managing VMs is based on an efficient mechanism to reach high availability under limited resources. Apart from this, how to study fault-tolerance on VMs and increase reliability are the other topics we want to address in this paper. In order to provide continuous availability for applications in case of server failure, a detection method is needed.

The virtualization fault tolerance (VFT) has three main phases: virtual machine migration policy, information gathering, and keeping services always available.

Virtual Machine Migration Policy: it enables DRA to make sure best performance in the distribution of the virtualization cluster.

Information Gathering: a detection mechanism is applied to retrieve all Hosts and check whether Hosts are alive or not. We detect states of the hosts with a ping command every five minutes by running a Linux schedule via "crontab."

Keeping Service Always Available: We assume that VM *m* is under the Heartbeat + DRBD mechanism, and Host *n* is going to become an unavailable physical machine. Once the Host *n* is shut down, if VM *m* is the secondary node, then it will be moved to an on-line Host and booted automatically. If VM *m* is the primary node, then the secondary node will replace VM *m* as the primary node immediately. Next pre-primary node will be booted on available host/hots and becomes secondary. In OpenNebula, command onevm is used to submit, control, and monitor VMs. It helps us control dead VMs and deploy them on other available physical hosts.

The workflow is shown in Fig. 9. However, there is a constraint that the number of physical hosts must be no less than three. It is the basic requirement to achieve VFT methodology and will be explained later.

This flow is implemented as one of the scheduled programs and deployed on the front-end. It is reasonable to enhance this function on the front-end of OpenNebula, because OpenNebula controls all VMs operations. Figure 10 depicts an example that explains how our VFT approach is triggered under single-failure events.

First, if Host A is shut down by unexpected events, in few minutes later, the front-end detects it and then triggers VFT. Next, the secondary node VM 2 becomes primary and handovers all services from preprimary, which is called as FAILOVER. Finally, VM 1 is booted on Host C automatically and becomes the secondary node, which is called as FAILBACK.

4 Experimental environment and results

4.1 Experimental environment

In our experimental environment, each server has same specifications. Table 3 lists CPU, memory, and storages capabilities of the servers. We measured the basic capability of their performance with known benchmarks. Next, we completed our experiment's data via Apache JMeter. We designed experiments for measuring server performance and throughputs as well. The well-known web application measurement performance tool, "Apache JMeter" [39], one of Apache projects, is open-sourced

UNCORRECTED PROOF

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

« SUPE 11227 layout: Small Extended v.1.1 file: supe1045.tex (Nadia) class: spr-small-v1.3 v.2013/11/19 Prn:2013/11/20; 16:09 p. 13/20»
« doctopic: OriginalPaper numbering style: ContentOnly reference style: basic»

AUTHOR'S PROOF

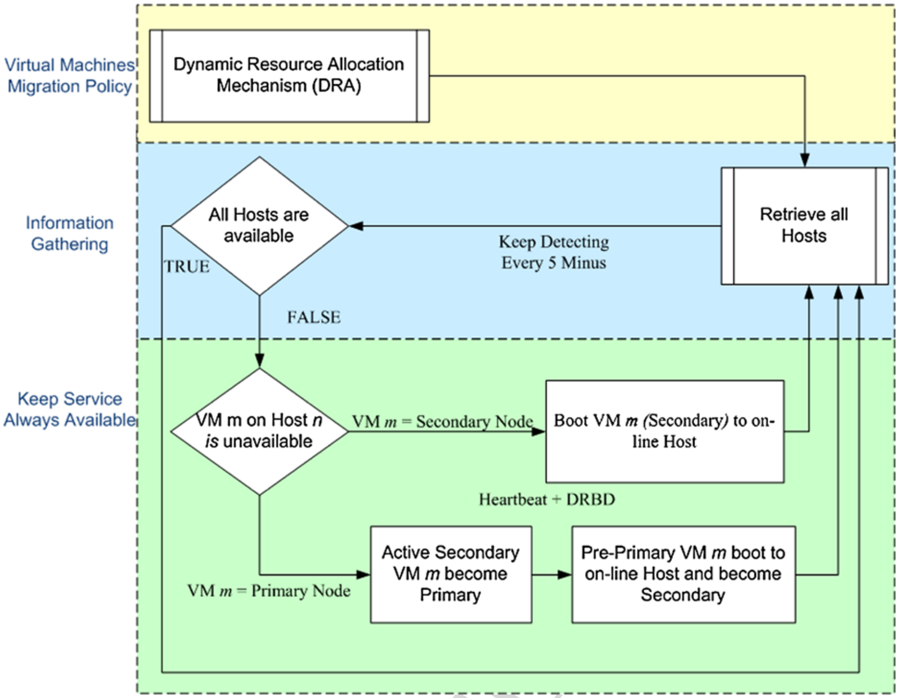

Fig. 9 The flow of virtualization fault tolerance

Table 3 Hardware specification of lab servers

No	Hardware lists					
	Model	Cores	CPU MHz	Disk (Giga)	Memory (Giga)	Comments
1	Intel(R) Core(TM) i7 CPU 860@2.80 GHz	4	2800	500	4	Front-End
2	Intel(R) Core(TM) i7 CPU 860@2.80 GHz	4	2800	500	4	Back-End
3	Intel(R) Core(TM) i7 CPU 860@2.80 GHz	4	2800	500	4	Back-End
4	Intel(R) Core(TM) i7 CPU 860@2.80 GHz	4	2800	500	4	Back-End

software and a 100 % pure Java desktop application designed to test functional behavior and measure performance. It was originally designed to test Web Applications but now has been expanded with other test functions.

4.2 Networking capability

In this section, we evaluate proposed architecture by studying the effect of virtualization of the worker nodes and physical hosts. In order to quantify different network

UNCORRECTED

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

« SUPE 11227 layout: Small Extended v.1.1 file: supe1045.tex (Nadia) class: spr-small-v1.3 v.2013/11/19 Prn:2013/11/20; 16:09 p. 14/20»
« doctopic: OriginalPaper numbering style: ContentOnly reference style: basic»

AUTHOR'S PROOF

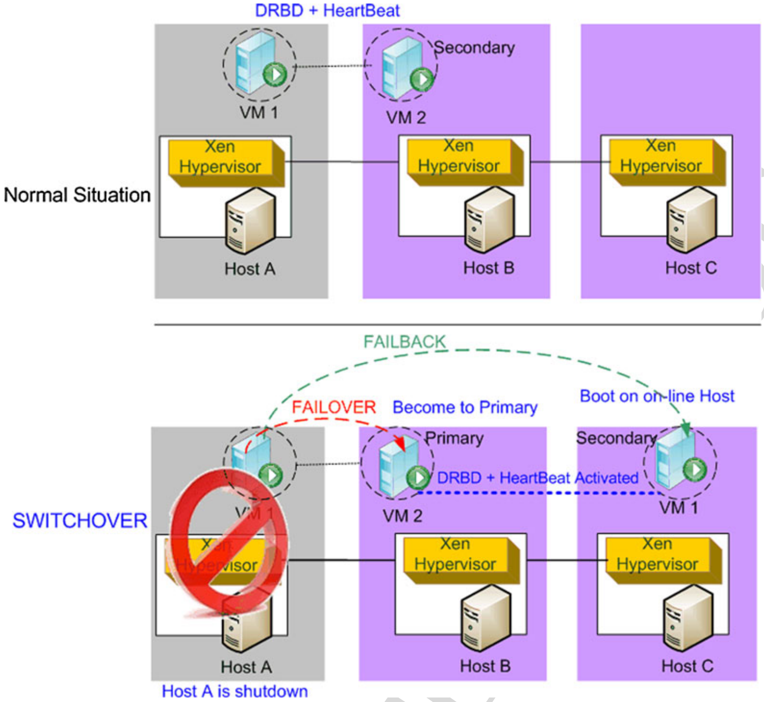

Fig. 10 The concept for how to trigger VFT

Table 4 Comparison of physical host and virtual machine networking performance

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

throughputs in local and remote nodes, we compare transfer times, using the HTTP protocol, and of different file sizes between the physical host and virtual machine. Under the same condition, Table 5 compares throughputs via the HTTP protocol with various file sizes and threads. A significant result is confirmed in Tables 4 and 5: the virtual machine performance is a little less than physical machine, but matches our expectation. Figures 11 and 12 show the network performance and throughputs for comparison of the physical host and virtual machine, respectively.

4.3 Life migration of virtual machine

We performed migration tests in an identical pair of server machines, each with eight i7-Core 2.8 GHz CPUs and 4 GB memory. The machines were connected via a

UNCORRECTED PROOF

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

« SUPE 11227 layout: Small Extended v.1.1 file: supe1045.tex (Nadia) class: spr-small-v1.3 v.2013/11/19 Prn:2013/11/20; 16:09 p. 15/20»
« doctopic: OriginalPaper numbering style: ContentOnly reference style: basic»

AUTHOR'S PROOF

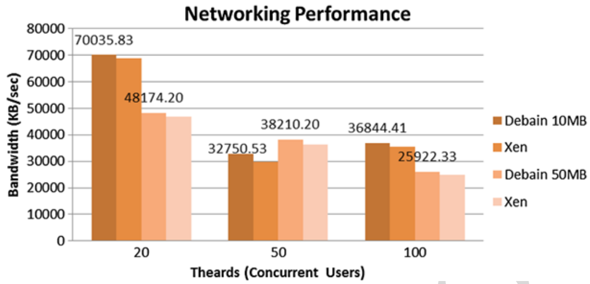

Fig. 11 Networking performance of physical host and virtual machine

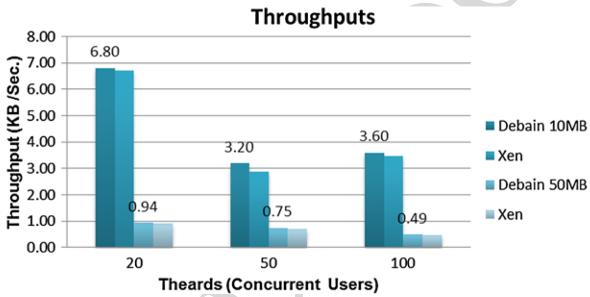

Fig. 12 Throughputs between physical host and virtual machine

Table 5 Throughputs between physical host and virtual machine

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

switched Gigabit Ethernet. Before migration, daemon required 1 G space on each host, thus, the maximum available memory space was 3 G for each host. There was only one VM on Host-A, and no VM on Host-B; the VM on Host-A used 1 G memory space. We migrated the VM from Host-A to Host-B. Figure 13 shows variations of memory usages of Host A, and Fig. 14 shows variations of memory usages of Host B, respectively.

4.4 Hadoop NameNode failover

In this experiment, we used settings listed in Tables 6 and 7, then built HDFS on a VM with one live node and 28.61 GB spaces. In this scenario, we monitored the HDFS failover while downloading. Another tool for this test is FUSE [41] that is chosen because it allows users to operate HDFS as a local disk.

When HDFS began downloading, the primary node (debian-ha1) was terminated as expected. The downloading action was disconnected after about 10–20 seconds; NameNode was then resumed on debian-ha2 automatically. This result only shows

UNCORRECTED

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

AUTHOR'S PROOF

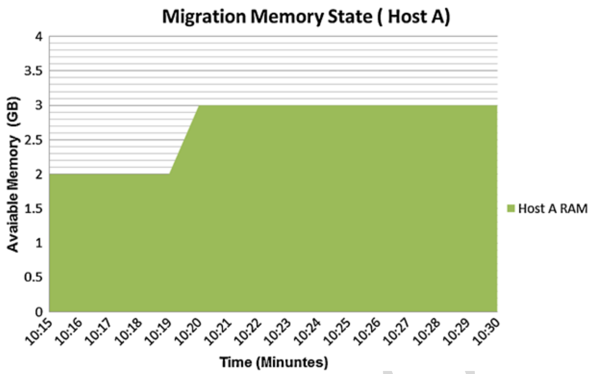

Fig. 13 Migration memory state of Host-A

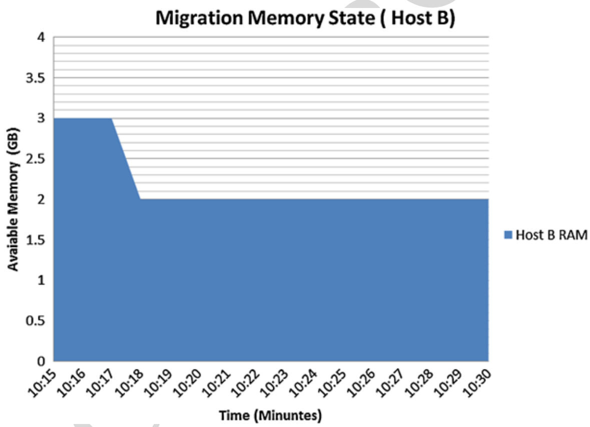

Fig. 14 Migration memory state of Host-B

our design is working on Active/Standby states. However, due to the metadata controlled by DRBD, the entire HDFS would not crash under unexpected system outages. It is a real enhancement for Hadoop NameNode because there are lots of issues related to NameNode failure problems after the unexpected system shutdown.

4.5 VFT experiment

In this scenario, we designed the experiment to validate if VMs automatically migrate when the host is off-line, as shown in Fig. 15. The Service IP is used for providing a service channel to external users. Users can access services through this IP, which also named VIP in DRBD speaking. Node 2 is the primary node, and Node 1 is the secondary. The difference between the primary and secondary node is that only the secondary node is allowed to take over the service if the primary node is down. Debian 1, Debian 2, and Debian 3 are the physical hosts. Node 2 lives on Debian 1; and the secondary node, Node 1, on Debian 2.

Figure 16 shows that after Debian 1 is disconnected, the Service IP does not terminate. Besides, only one packet is lost during the time when the failover behavior is enabled. The reason is that the entire system is under the VFT mechanism in which the secondary node can immediately replace the primary node if it is down. Finally,

UNCORRECTED

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

AUTHOR'S PROOF

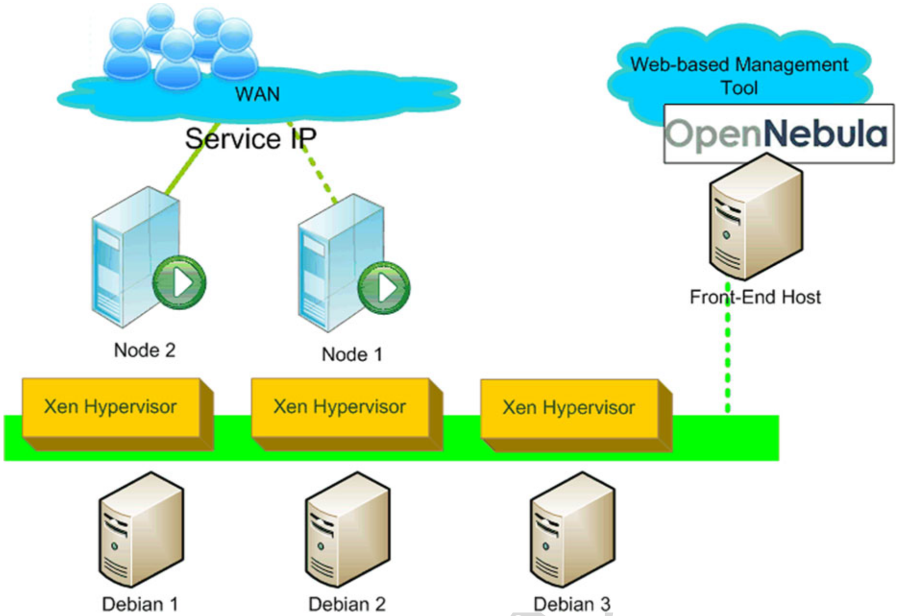

Fig. 15 VFT experiment environments

Table 6 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 7 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

UNCORRECTED

Node 2 is rebooted automatically in about 5 to 7 minutes and becomes secondary. The benefit of VFT mechanism is to obtain a shortest downtime interval. Although we cannot guarantee the integrity of data during the downtime period, we still can reduce the downtime interval to provide a low-cost solution for enterprises.

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

« SUPE 11227 layout: Small Extended v.1.1 file: supe1045.tex (Nadia) class: spr-small-v1.3 v.2013/11/19 Prn:2013/11/20; 16:09 p. 18/20»
« doctopic: OriginalPaper numbering style: ContentOnly reference style: basic»

AUTHOR'S PROOF

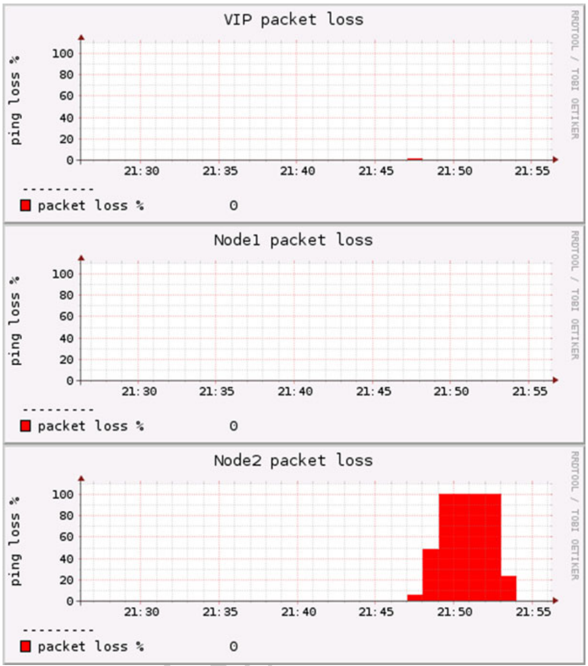

Fig. 16 Measurements of ping loss

5 Conclusions

High-availability is achieved in Hadoop NameNode Active-Standby architecture. Under this architecture, the service can be failed over when the primary node is failed. The most valuable improvement in this paper is that by keeping at least three physical hosts available, then primary and secondary nodes will always exist. Therefore, there are four main key features in this work: the first is Xen Hypervisor; the second, OpenNebula; the third, DRBD with Heartbeat component; and the last, the VFT mechanism. Each component is important and indispensable in our architecture. Systems with continuous availability mean comparatively higher priced, and most have carefully implemented with special designs that eliminate any single point of failure and allow online hardware, network, operating system, middleware, and application upgrades, patches, and replacements. However, the future goal of this paper is to extend our fault-tolerance work beyond failure management in order to enhance resources utilization efficiency of virtualization clusters.

Acknowledgements This work is supported in part by the National Science Council, Taiwan R.O.C., under grants no. NSC 101-2221-E-029-014, NSC 101-2622-E-029-008-CC3, and NSC 101-2218-E-029-004.

UNCORRECTED

References

1. Chaudhary V, Minsuk C, Walters JP, Guercio S, Gallo S (2008) A comparison of virtualization technologies for HPC. In: 22nd international conference on advanced information networking and applications, AINA 2008, pp 861–868

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF

2. Rafael M-V, Ruben SM, Ignacio ML (2009) Elastic management of cluster-based services in the cloud. In: Proceedings of the 1st workshop on automated control for datacenters and clouds, Barcelona, Spain. ACM, New York, pp 19–24
3. Engelmann C, Scott SL, Leangsuksun C, He X (2008) 8th IEEE international symposium on symmetric active/active high availability for high-performance computing system services: accomplishments and limitations. In: Cluster computing and the grid, CCGRID '08, pp 813–818
4. Turner D, Xuehua C (2002) Protocol-dependent message-passing performance on Linux clusters. In: IEEE international conference on cluster computing, proceedings, pp 187–194
5. Xen (2013) Available: http://www.xen.org/. Accessed 3 June 2013
6. Hadoop (2013) Available: http://hadoop.apache.org. Accessed 3 June 2013
7. Ning C, Zhong-hai W, Hong-zhi L, Qi-xun Z (2010) Improving downloading performance in hadoop distributed file system. J Comput Appl 30
8. Grossman RL, Gu Y, Sabala M, Zhang W (2009) Compute and storage clouds using wide area high performance networks. Future Gener Comput Syst 25:179–183
9. Shafer J, Rixner S, Cox AL (2010) The hadoop distributed filesystem: balancing portability and performance. In: IEEE international symposium on performance analysis of systems & software (ISPASS), White Plains, NY, pp 122–133
10. Mackey G, Sehrish S, Jun W (2009) Improving metadata management for small files in HDFS. In: IEEE international conference on cluster computing and workshops, CLUSTER'09, pp 1–4
11. Cloudera. Available: http://www.cloudera.com
12. Xuhui L, Jizhong H, Yunqin Z, Chengde H, Xubin H (2009) Implementing WebGIS on hadoop: a case study of improving small file I/O performance on HDFS. In: IEEE international conference on cluster computing and workshops, CLUSTER'09, pp 1–8
13. White T (2010). Hadoop: The Definitive Guide, 2nd edn
14. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008) Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst 26:1–26
15. Ghemawat S, Gobioff H, Leung S-T (2003) The Google file system. Oper Syst Rev 37:29–43
16. Engelmann C, Scott SL, Leangsuksun C, He X (2006) Active/active replication for highly available HPC system services. In: The first international conference on availability, reliability and security, ARES 2006, p 7
17. Fei-fei L, Xiang-zhan Y, Gang W (2009) Design and implementation of high availability distributed system based on multi-level heartbeat protocol. In: IITA international conference on control, automation and systems engineering, CASE 2009, pp 83–87
18. Walters J, Chaudhary V (2009) A fault-tolerant strategy for virtualized HPC clusters. J Supercomput 50:209–239
19. Vargas E (2000) High availability fundamentals. Sun Microsystems, Santa Clara
20. Vallee G, Engelmann C, Tikotekar A, Naughton T, Charoenpornwattana K, Leangsuksun C, Scott SL (2008) A framework for proactive fault tolerance. In: Third international conference on availability, reliability and security, ARES 08, pp 659–664
21. Ang C-W, Tham C-K (2007) Analysis and optimization of service availability in a HA cluster with load-dependent machine availability. IEEE Trans Parallel Distrib Syst 18:1307–1319
22. Dejan M, Liorente LM, Montero RS (2011) OpenNebula: a cloud management tool. IEEE Internet Comput 15:11–14
23. Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D (2009) 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
24. Sempolinski P, Thain D (2010) A comparison and critique of eucalyptus, OpenNebula and Nimbus. In: IEEE second international conference on cloud computing technology and science (CloudCom), pp 417–426
25. Yang C-T, Cheng H-Y, Chou W-L, Kuo C-T (2011) A dynamic resource allocation model for virtual machine management on cloud. In: Symposium on cloud and service computing
26. Piedad F, Hawkins M (2001) High availability, design, techniques and processes. Prentice-Hall, New York
27. DRBD Official Site (2013) Available: http://www.drbd.org. Accessed 3 June 2013
28. Heartbeat—Linux High Availability (2013) Available: http://linux-ha.org/wiki/Heartbeat. Accessed 3 June 2013
29. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the art of virtualization. Oper Syst Rev 37:164–177

UNCORRECTED PROOF

« SUPE 11227 layout: Small Extended v.1.1 file: supe1045.tex (Nadia) class: spr-small-v1.3 v.2013/11/19 Prn:2013/11/20; 16:09 p. 20/20»
« doctopic: OriginalPaper numbering style: ContentOnly reference style: basic»

AUTHOR'S PROOF

30. Hagen Wv (2008) Professional Xen Virtualization
31. Yang C-T, Tseng C-H, Chou K-Y, Tsaur S-C (2009) Design and implementation of a virtualized cluster computing environment on Xen. Presented at the second international conference on high performance computing and applications, HPCA
32. Nagarajan AB, Mueller F, Engelmann C, Scott SL (2007) Proactive fault tolerance for HPC with Xen virtualization. Presented at the proceedings of the 21st annual international conference on supercomputing, Seattle, Washington
33. Montero RS, Moreno-Vozmediano R, Llorente IM (2011) An elasticity model for high throughput computing clusters. J Parallel Distrib Comput 71:750–757
34. Sotomayor B, Montero RS, Llorente IM, Foster I (2009) Virtual infrastructure management in private and hybrid clouds. IEEE Internet Comput 13
35. OpenVZ. Available: http://wiki.openvz.org/Main_Page
36. OpenNebula. Available: http://www.opennebula.org
37. Hai Z, Kun T, Xuejie Z (2010) An approach to optimized resource scheduling algorithm for open-source cloud systems. In: Fifth annual ChinaGrid conference (ChinaGrid), pp 124–129
38. Chen Q, Zhang D, Guo M, Deng Q, Guo S (2010) SAMR: a self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In: 10th IEEE international conference on computer and information technology, pp 2736–2743
39. Apache JMeter. Available: http://jakarta.apache.org. Accessed 3 June 2013
40. HPCC. Available: http://icl.cs.utk.edu/hpcc. Accessed 3 June 2013 <uncited>
41. MountableHDFS. Available: http://wiki.apache.org/hadoop/MountableHDFS. Accessed 3 June 2013

UNCORRECTED PROOF

Journal ID: 11227, Article ID: 1045, Date: 2013-11-21, Proof No: 1, UNCORRECTED PROOF